National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Machine Learning-based Surrogate models for Uncertainty Quantification in CFD

Diploma Thesis

Dionysios Bakis

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Acknowledgments

I want to express my deepest gratitude to Professor Kyriakos Giannakoglou for his constant guidance and support throughout my Diploma Thesis. His deep knowledge, thoughtful advice, and genuine encouragement have made a massive difference in my work and overall learning experience. I feel lucky to have had the chance to learn from him, and I have grown a lot thanks to his mentorship and the invaluable insights he has shared with me. He has been a continuously inspiring figure throughout my studies.

Secondly, I want to express my sincere appreciation to Dr. Marina Kontou and Dr. Varvara Asouti, who provided me with invaluable knowledge and support whenever I needed it. Moreover, I want to thank every member of the PCOpt/NTUA I met, as everyone was very respectful and helpful regarding any interaction I had there.

I am deeply grateful to my family for their tireless support and belief in me.

I'm thankful to all my friends who have accompanied me on the journey until today, creating unforgettable memories and being there for me. Lastly, I have to give special thanks my dear friend Theodoros Papaiakovou who always believed in me and provided me with essential inspiration in the fields of Engineering and Artificial Intelligence.

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Machine Learning-based Surrogate models for Uncertainty Quantification in CFD

Diploma Thesis

Dionysios Bakis

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Abstract

This Diploma Thesis is in the area of Aerodynamic Shape Optimization under Uncertainties, and, in particular, it explores the adequacy of Feedforward Fully Connected Deep Neural Networks as surrogates of the high-fidelity, yet very expensive, Computational Fluid Dynamics solver for the task of quantifying the effects of uncertainty (Uncertainty Quantification, UQ). This is the key process in search for a well performing design that is less sensitive to the presence of uncertainties (Robust Design), instead of just the best-performing design point.

Uncertainties must be taken into account whenever optimization is carried out. For example, in Mechanical Engineering, minor fluctuations in model parameters can yield sub-optimal performances. In conjunction with the evolution of computing systems, the use of methods that take uncertainties into account increases the reliability of the outcome of an optimization process.

In most cases, UQ requires the computation of a stochastic model's mean and variance. Here, UQ is carried out using Monte Carlo and Polynomial Chaos Expansion coupled with the fluid solver or a surrogate model for two aerodynamic problems involving transitional flows: an isolated airfoil and an isolated wing. The flows are simulated with the in-house PUMA software of the Parallel CFD & Optimization Unit solving the Reynolds-Averaged Navier-Stokes equations along with the Spalart-Allmaras turbulence model, and the $\gamma - \tilde{Re}_\theta$ transition model. Cases with uncertainties related to coefficients that appear in the $\gamma - \tilde{Re}_\theta$ transition model are studied. The UQ methods, in general, require repetitive calls to the analysis code, which renders them prohibitively expensive, especially, whenever CFD software is involved. For this reason, the development of surrogate models aims to accelerate the optimization process by greatly reducing the computational cost. On the other side,

surrogate models perform computations of lower fidelity than those obtained from the expensive CFD tool. This trade-off is being investigated in the two aerodynamic problems.

The DNNs' hyperparameters are tuned manually and also, by using evolutionary algorithms. Lastly, the involvement of two very promising techniques (Stacking Ensemble and Feature Selection) is examined, aiming to check how the prediction accuracy can further be improved as well as how surrogate models with fewer input features perform compared to those used so far.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Υποκατάστατα Μοντέλα βασισμένα στη Μηχανική Μάθηση για Ποσοτικοποίηση Αβεβαιοτήτων στην Υπολογιστική Ρευστοδυναμική

Διπλωματική Εργασία

Διονύσιος Μπακής

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Περίληψη

Αυτή η Διπλωματική Εργασία υλοποιείται στην περιοχή της βελτιστοποίησης μορφής σωμάτων με αεροδυναμικά κριτήρια, λαμβάνοντας υπόψη αβεβαιότητες. Ειδικότερα, διερευνά την επάρκεια των πλήρως συνδεδεμένων βαθιών νευρωνικών δικτύων πρόσθιας τροφοδότησης ως υποκατάστατων του υψηλής πιστότητας, αλλά πολύ ακριβού, επιλύτη Υπολογιστικής Ρευστοδυναμικής για την ποσοτικοποίηση των επιπτώσεων της αβεβαιότητας (Ποσοτικοποίηση Αβεβαιότητας). Αυτή είναι η βασική διαδικασία στην αναζήτηση της βέλτιστης αεροδυναμικής μορφής που μπορεί να θεωρηθεί λιγότερο ευαίσθητη στην επίδραση αβεβαιοτήτων (Στιβαρός Σχεδιασμός ή UQ). Αντί για το σημείο σχεδιασμού με τις καλύτερες επιδόσεις, ο Στιβαρός Σχεδιασμός επικεντρώνεται στον εντοπισμό μιας λύσης η οποία ελαχιστοποιεί τις ανεπιθύμητες συνέπειες των αβεβαιοτήτων.

Οι αβεβαιότητες πρέπει να λαμβάνονται υπόψη κάθε φορά που πραγματοποιείται βελτιστοποίηση. Για παράδειγμα, στη Μηχανολογία, μικρές διακυμάνσεις στις παραμέτρους ενός μοντέλου μπορούν να αποδώσουν επιδόσεις που απέχουν από τη βέλτιστη. Σε συνδυασμό με την εξέλιξη των υπολογιστικών συστημάτων, η χρήση μεθόδων που λαμβάνουν υπόψη αβεβαιότητες, αυξάνει την αξιοπιστία του αποτελέσματος μιας διαδικασίας βελτιστοποίησης.

Στις περισσότερες περιπτώσεις, η UQ απαιτεί τον υπολογισμό του μέσου όρου και της διακύμανσης ενός στοχαστικού μοντέλου. Εδώ, η UQ πραγματοποιείται με τη χρήση Monte Carlo και της μεθόδου του αναπτύγματος πολυωνυμικού χάους σε συνδυασμό με τον επιλύτη της Υπολογιστικής Ρευστοδυναμικής (CFD) ή ένα υποκατάστατο μοντέλο για δύο αεροδυναμικά προβλήματα που αφορούν μεταβατικές ροές: μια μεμονωμένη αεροτομή και μία μεμονωμένη πτέρυγα. Οι ροές προσομοιώνονται με το οικείο

λογισμικό PUMA της Μονάδας Παράλληλης Ρευστοδυναμικής & Βελτιστοποίησης του ΕΜΠ, επιλύοντας τις Reynolds-Averaged Navier-Stokes εξισώσεις μαζί με το Spalart-Allmaras μοντέλο τύρβης και το μοντέλο μετάβασης $\gamma - \tilde{R}e_\theta$. Μελετώνται περιπτώσεις με αβεβαιότητες που σχετίζονται με συντελεστές που εμφανίζονται στο μοντέλο μετάβασης $\gamma - \tilde{R}e_\theta$. Οι μέθοδοι UQ, γενικά, απαιτούν επαναλαμβανόμενες κλήσεις στον κώδικα ανάλυσης, γεγονός που τις καθιστά απαγορευτικά ακριβές, ειδικά, όταν εμπλέκεται λογισμικό CFD. Για το λόγο αυτό, η ανάπτυξη υποκατάστατων μοντέλων αποσκοπεί στην επιτάχυνση της διαδικασίας βελτιστοποίησης, μειώνοντας σημαντικά το υπολογιστικό κόστος. Από την άλλη πλευρά, τα υποκατάστατα μοντέλα πραγματοποιούν υπολογισμούς χαμηλότερης πιστότητας από εκείνους που πραγματοποιούνται με το ακριβό εργαλείο CFD. Αυτό το αντιστάθμισμα διερευνάται στα δύο αεροδυναμικά προβλήματα.

Η ρύθμιση των υπερπαραμέτρων των DNN πραγματοποιείται χειροκίνητα καθώς επίσης, και με τη χρήση εξελικτικών αλγορίθμων. Τέλος, διερευνάται η εμπλοκή δύο πολύ υποσχόμενων τεχνικών (Stacking Ensemble και Feature Selection), με στόχο να ελεχθεί πώς μπορεί να βελτιωθεί περαιτέρω η ακρίβεια πρόβλεψης, καθώς και απόδοση υποκατάστατων μοντέλων με λιγότερες μεταβλητές εισόδου.

Nomenclature

CFD	Computational Fluid Dynamics
NTUA	National Technical University of Athens
PCOpt	Parallel CFD & Optimization unit
GBM	Gradient Based Method
SOO	Single Objective Optimization
MOO	MultiO bjective Optimization
RDO	Robust Design Optimization
UQ	Uncertainty Quantification
MC	Monte Carlo
PCE	Polynomial Chaos Expansion
niPCE	Non-Intrusive PCE
GPU	Graphical Processing Unit
RANS	Reynolds-Averaged Navier-Stokes
ASO	Aerodynamic Shape Optimization
AoA	Angle of Attack
LHS	Latin Hypercube Sampling
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
LR	Linear Regression
ANN	Artificial Neural Network
RBFN	Radial Basis Function Network
DNN	Deep Neural Network
DFSS	Design for Six Sigma
TP	Training Patterns
SA	Spalart Allmaras
PDF	Probability Density Function

Contents

Chapter 1

Fundamentals of Optimization Under Uncertainties

As technology evolution continues its accelerating ascent and computers become more and more powerful, there is an increasingly growing interest in how existing real-world solutions can be optimized or improved. Moreover, due to this availability of computational power, incorporating sources of uncertainty in optimization problems has also become feasible. The latter has led to establishing the well-known Robust Design Optimization methods. The introductory chapter assesses the core principles of optimization that lay the groundwork for this thesis, along with the main obstacles and turnarounds that emerge when considering sources of uncertainty.

1.1 Introduction to Optimization

The term "optimization" is closely associated with the word "objective" and refers to a process used across various fields and industries to track the best solution to a specific problem. On the other hand, "improvement" refers to the process of finding a solution that is simply better than the current one. Since both processes aim to enhance outcomes, the latter also falls under the umbrella of optimization. More precisely, optimization can be applied to any outcome influenced by N, the number of controlled causes, the so-called optimization or design variables ($\vec{b} \in R^N$). Therefore, since every cause-and-effect relationship can be explicitly or approximately described by a mathematical formulation, the term "objective function" is introduced. This function effectively describes the correlation between a Quantity of Interest (QoI) and some inputs in any problem. In terms of seeking the "best

solution", the optimization process aims to find the global maximum or minimum of the objective function. (The critical distinction with improvement is that the last does not necessarily seek the global extrema; instead, the process concludes upon reaching some local extremum). For example, in a car optimization problem, the goal could be the minimization of the aerodynamic drag force, the minimization of the total cost of the car, or the maximization in sales. These objectives can coexist in the optimization process, leading to a problem of many objective functions. This distinction leads to the two optimization subclasses: single and multi-objective optimization (SOO and MOO) [1, 2].

Every optimization algorithm iterates until it converges. The process begins with the initialization of the optimization variables. Then, the computation of the objective function takes place. Based on this information and the use of a selected optimization method, adjustments are made to the design variables' values for the next iteration. This cycle repeats until there is no further improvement in the objective function, according to the user-defined termination criterion. Eventually, the optimal design vector is known. It's worth noting that the optimization process becomes "constrained" when there are limitations related to the problem at hand. For example, limitations will possibly exist in the design variables' search spaces.

1.2 Optimization Methods and Costs

In the preceding process, the total computational cost is strongly tied to the choice of the optimization method. This choice is significant as it determines the cost for each cycle, which is then multiplied by the total number of cycles needed. Therefore, the choice of the optimization method directly impacts the overall computational cost.

Optimization methods pertain to the adjustments of the design variables, utilizing the objective function's calculation. They can be stochastic (gradient-free), deterministic (gradient-based) or a combination of the two.

Stochastic methods treat the objective function as a black box and use only its value. Hence, the cost of every cycle is determined by the price of the objective function's calculation multiplied by the times it needs to be evaluated. They employ stochastic searching techniques in the hunt for the best solution and can be applied to any problem, even when the formulation of the objective function is not accessible. Due to their inherent stochasticity, they don't get trapped into local extrema, if the search algorithm is free to run as much as needed. Their main disadvantage is that they demand a considerable number of objective function evaluations to converge, compared to deterministic optimization methods [1, 3]. A stochastic optimization method example is the evolutionary algorithms, where the choice of the best design vector occurs based on bio-inspired procedures that emulate natural selection. The latter is the optimization method utilized in this thesis through the Evolutionary

Algorithms SYstem - EASY software [4], developed by the PCOpt/NTUA.

On the other hand, deterministic methods aim at a faster convergence, thanks to the gradient information, but this speed comes with a trade-off. There is a risk of getting trapped in a local extremum. An example is the steepest descent method.

$$\vec{b}^{\text{new}} = \vec{b}^{\text{old}} - \eta \frac{\partial F(\vec{b})}{\partial \vec{b}} \tag{1.1}$$

where F is the objective function and η is a user-defined tuning parameter, called learning rate. The computation of the sensitivity derivatives, corresponds to part of the cost in each optimization cycle.

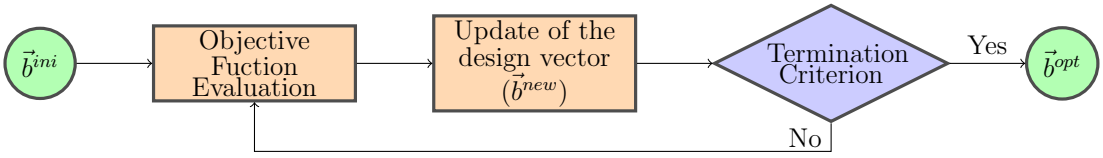

Figure 1.1: *Conventional Optimization flowchart. First, the design variables are initialized. Next, the objective function is computed by the primal solver as many times as the selected optimization method indicates, and the values of the design variables are updated. When the termination criteria is met, the process stops and the optimal design vector is obtained. Figure adapted from [5].*

1.3 Optimization Under Uncertainties

Almost all real-world problems, such as shape optimization, are subjected to some level of uncertainty. It can be related to stochastic perturbations that influence the environment, the design parameters, or the evaluation of the system [6]. These uncertainties involved in the optimization procedure can lead to results that deviate significantly from the "optimal" ones. For example, as stated in [6] *"even if one were able to map the model optimum to the true optimum, one might not be able to build the true optimum either because of manufacturing uncertainties or because the required precision during this manufacturing stage would be too costly".*

As a result, "Optimization Under Uncertainties" or "Robust Design Optimization" is introduced. Within this thesis, the focus is on uncertainties related to the system's environment parameters. Thus, a classification between the system's input variables is conducted. As mentioned above, there are the N design variables ($\vec{b} \in R^N$), which are under the designer's control. There are also the M environmental, robust, or uncertain variables ($\vec{c} \in R^M$), which affect the system's environment and are subjected to some degree of stochasticity. The environmental variables are out of the designer's control. So, the objective function takes the form $F = F(\vec{b}, \vec{c})$.

In cases uncertainties coincide with some of the design variables, they can be handled as in the following example. In real-world manufacturing, achieving the "optimal" length b_κ of a mechanical component is often challenged by geometric tolerances, which introduce inherent uncertainties into the design dimensions. If this source of uncertainty is decided to be taken into account, the length will be expressed as $b_\kappa + \Delta b_\kappa$. The stochastic length's perturbation will be processed as a new environmental variable and will be added to the $\vec{c}$ vector.

Following an RDO statistical approach [7], it must be highlighted that, in order to model the uncertain variables, the user has to assume that each one of them follows a known probability density function (PDF). In this Diploma Thesis, an approach based on the well-known Design for Six Sigma (DFSS) was employed [8, 9]. While this is not obligatory, all uncertain variables were assumed to be normally distributed around their mean values within the interval $[\mu_i - 3\sigma_i, \mu_i + 3\sigma_i]$, (three sigma). The latter inherits from the fact that when a stochastic quantity follows the normal distribution, 99.73% of the time will fall inside a "six sigma" interval around its mean value.

The goal of RDO is to achieve a solution that is not only optimal but also remains insensitive to variations in its input uncertain variables. In figure 7.4, the maximization of a QoI named F is aimed. The difference between true optimal and robust optimal solution can be observed.

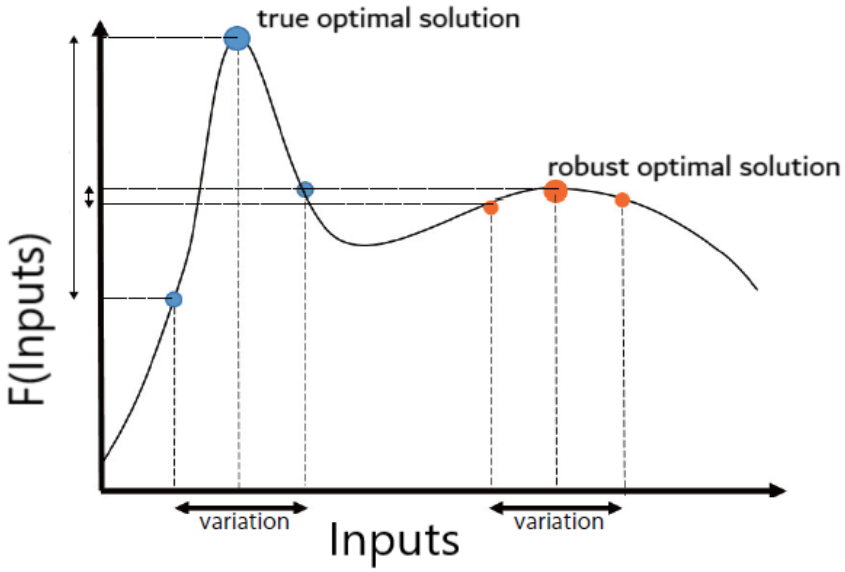

Figure 1.2: *True optimal vs robust optimal solution. If the design is conducted for the true optimal, meaning the function's global maximum, slight uncertainties in the input variables can lead to highly undesirable outcomes. Hence, identifying the robust optimal means finding a valley-like region within the output space. In this case, variations in the inputs result in negligible changes in the output value of F, making the design far more reliable. Figure adapted from [10]*

Aiming to find the robust optimal, a new objective function must be defined. Considering as QoI the F function of figure 7.4, the interest passes to the maximization of the F's mean value μ_F. Besides μ_F, a second commonly added criterion is also the minimization of F's variance or standard deviation σ_F. Thus, a common approach that takes into account the trade-off between the mean and the standard deviation is to optimize their weighted sum:

$$F_R = w_0\mu_F + w_1\sigma_F$$

This new objective function is known as the Robustness Metric. The user-defined weight values dictate the priority of the aforementioned quantities in the optimization process.

During the optimization under uncertainties, for each candidate design vector $\vec{b}$, the F's mean and variance must be computed. This process is referred to as Uncertainty Quantification (UQ), and there are various UQ methods, stochastic and deterministic, that can be utilized for it. For example, a deterministic one is the Method of Moments [11] which computes the μ_F and σ_F by firstly computing the first and second derivatives of F w.r.t the uncertain variables. The UQ methods assessed in this Thesis, in the two aerodynamic cases, are the simplest yet most expensive Monte Carlo (MC) method and the Polynomial Chaos Expansion (PCE). MC and PCE are presented in chapter 4.

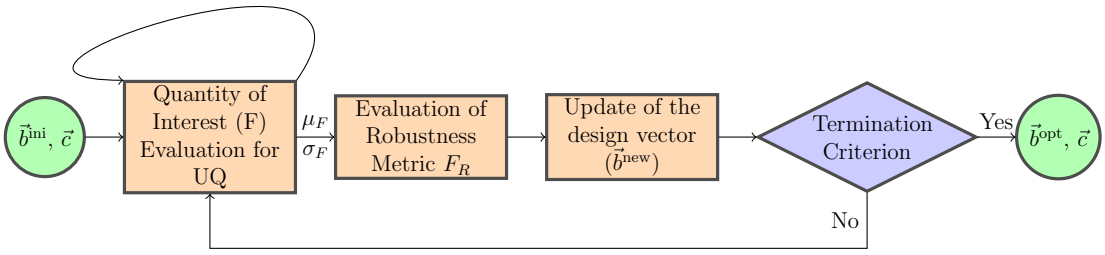

Figure 1.3: *Robust Design Optimization Flowchart. First, the design variables are initialized. Next, depending on the chosen UQ method μ_F and σ_F are being computed and their weighted sum is obtained (F_R). The latter is the only difference between RDO and conventional optimization. After UQ, the problem is the same as conventional optimization, but instead of F, the chosen optimization method is dealing with F_R. Thus, the design variables are updated, and the process continues until the termination criterion is met. Figure adapted from [5].*

1.4 AI Integration to CFD and UQ

Using AI to predict flows or flow quantities, the concept of surrogate modeling for CFD has seen significant development over the recent years, offering substantial computational savings and improvements in simulation time [12, 13].

Overall, these advancements signify a shift towards more data-driven approaches in engineering simulations. AI enhances traditional methods by reducing computational overhead and enabling more frequent simulations without the associated high costs. Chapter 2 delves into AI theory and presents AI algorithms used herein.

The term Time Unit (TU) which is often used in the Thesis is defined as follows: one TU corresponds to the computational cost of simulating the flow solution, using the high-fidelity CFD tool.

Regarding the use of AI in the present study, the key difference between a conventional optimization and an RDO algorithm must be highlighted: in RDO, the UQ must be carried out for each candidate solution, namely for each candidate design vector. All UQ methods require some, let's say X, evaluations of the QoI to compute μ and σ. The latter translates to X TUs only for the UQ. Thus, in Aerodynamic Shape Optimization (ASO) problems, where uncertainties are considered, X additional calls to the CFD for every candidate solution are needed. Therefore, for performing optimization under uncertainties, the total optimization cost (the cost of the same problem when optimized conventionally) is multiplied by X. Monte Carlo, for example, usually requires more than 1000 samples. The latter makes the use of MC, coupled with the CFD, prohibited for UQ in general ASO problems.

Except from developing more efficient UQ methods like PCE, a process far less costly than the MC, this significant increase in computational cost has led to the employment of surrogate AI models instead of the CFD, for UQ, as evaluation tools. In general, these models are trained to predict the outputs of the CFD simulations. They are of a much lower total cost, yet with the expense of reduced accuracy. The CFD is used to produce a DB containing the system inputs and outputs, the so-called Training Patterns (TP). The created DB is then used to train the AI model. Considering the usage and training costs of the currently developed AI models as negligible, the total cost of building the surrogate is predominantly determined by the creation of the DB, which is far less than the cost of the CFD evaluations needed for the optimization. Related research topics can be found in [14, 15, 16, 17] where surrogate ML models such as Artificial Neural Networks (ANNs) were used as CFD-surrogates, specifically for UQ.

1.5 Thesis Outline

The structure of this Diploma Thesis is presented:

- **Chapter 2:** An introduction to the field of Artificial Intelligence (AI) with a focus on function approximation. Fundamental Machine Learning (ML) algorithms are presented, and a brief explanation of how they can be combined to form ensemble models is provided. Significant emphasis is placed on the concept of Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs) along with how they can be tuned and trained for optimal performance.
- **Chapter 3:** The equations solved by the Computational Fluid Dynamics (CFD) code regarding the assessed aerodynamic problems. More precisely, the mean flow equations, turbulence, and transition models are presented in brief.
- **Chapter 4:** The presentation of the UQ methods employed in this work, mainly focused on the mathematics behind Non-Intrusive PCE (niPCE) coupled with either Gauss Quadrature Integration or Regression, methods for the computation of the PCE coefficients.
- **Chapter 5:** The DNN surrogate models are investigated and employed for the task of UQ in the case of NLF(1)-0416 isolated airfoil. Utilizing the same testing Database (DB), a comparison between DNNs and Radial Basis Function Networks (RBFNs) is made. A combination of the two is also checked through the stacking ensemble technique, aiming to improve the accuracy further. Last, DNNs are trained using feature selection to retain only the most relevant features as inputs, and their performance is discussed.
- **Chapter 6:** The development of surrogate DNN models, primarily using EASY for hyperparameter tuning, in order to perform UQ in the case of the flow around the ONERA M6. In this case, different DNNs are directly employed for UQ, and their performance is checked using solely the UQ results. Additionally, a brief study on the predicting abilities of DNNs, KNNs, and SVRs is conducted.

Chapter 2

Artificial Intelligence

2.1 Introduction to Artificial Intelligence

The visible impact of AI in the day to day life provided much inspiration for this thesis. Nowadays, large-scale and "free-to-use" virtual assistants can browse the internet and provide the user with specific information of any type, such as creating pictures, programs, and even videos based on single prompts [18, 19, 20, 21, 22]. Medical AI tools can perform diagnostic imaging evaluations [23]. Personalized recommendation systems capitalize on all types of preferences of every social media user, based on the actions he takes every second he uses the corresponding platforms. The applications above are a potential everyday interaction with AI for many people in almost every country. AI tools are becoming increasingly versatile and integrated to some degree in almost every industry. However, AI is a relatively hard term to be explicitly defined. For example, what exactly is *"Intelligence"*? A simple answer stated in [24] is that: *"Intelligence is the ability to process information such as we can use it to inform a future decision or action that we take"*. Yet, AI can be defined as systems that can perform tasks that require Human Intelligence. These tasks include problem-solving, perception, learning, understanding natural language, and the ability to move and manipulate objects. This thesis focuses on two well-known subfields of AI: **Machine Learning (ML)** and **Deep Learning (DL)**. In particular, DL is also a subfield of ML. These categories are visualized in figure 2.1.

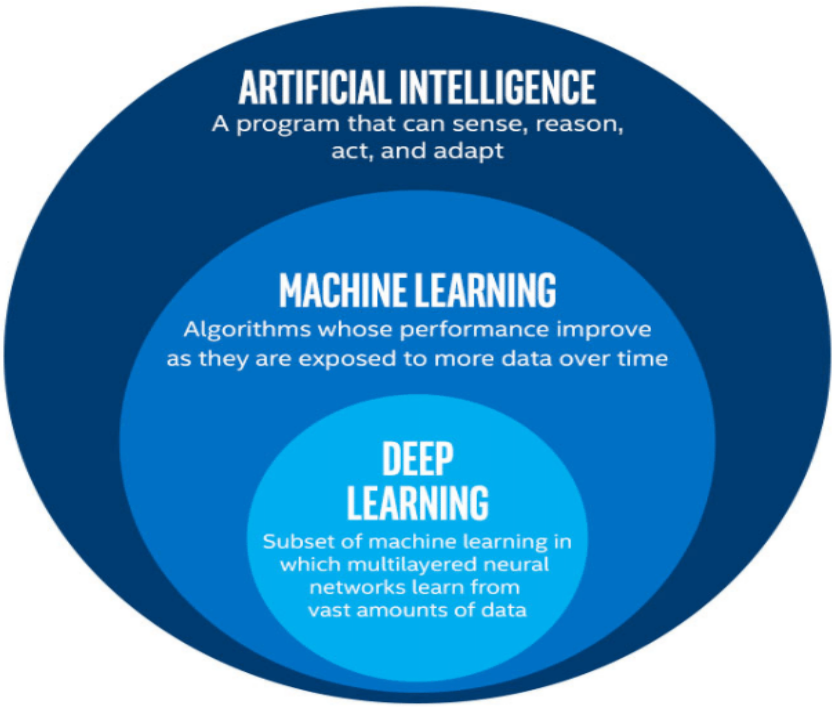

Figure 2.1: *AI subfields, ML and DL. Figure from [25]*

2.2 Machine Learning

ML is the subfield of AI where algorithms can learn from data and, thus, improve their performance. Going through the age of Big Data, it is clear why ML can be very powerful. It is worth noting that once a model is trained, it can execute its task numerous times with relatively low additional costs regarding computational resources. There are three main learning categories.

- **Supervised Learning** is a process where the model is given some labeled inputs i.e inputs with their corresponding outputs. It is similar to when a child is under the supervision of a teacher. At the end of the training process, the model can give correct answers to unseen data. There are two main tasks underlying this form of learning. Classification is the first one, where the output is one of a finite set of values, and the goal is to categorize the inputs into classes. The second one is called regression, which refers to when a model outputs numbers. In fact, in classification, there is always an underlying regression taking place internally to calculate the probability of each sample to belong to each class.

- **Unsupervised Learning** is the category where the model tries to identify underlying patterns in unlabelled data received as inputs. In this one, no explicit

feedback is provided to the model. The most common task of this category is called clustering, which is about the definition of clusters that include inputs similar to each other. In recommendation systems, e.g., clustering is used to group customers based on the data they have provided to identify segments of users with similar tastes or needs.

- **Reinforcement Learning** is inspired by learning methods based on reward and punishment. It involves an agent who interacts with an environment, takes actions, and is rewarded or punished at some point. Over time, the agent aims to develop a policy that maximizes the cumulative reward. A typical example of this category is when computers are trained to master a video game autonomously.

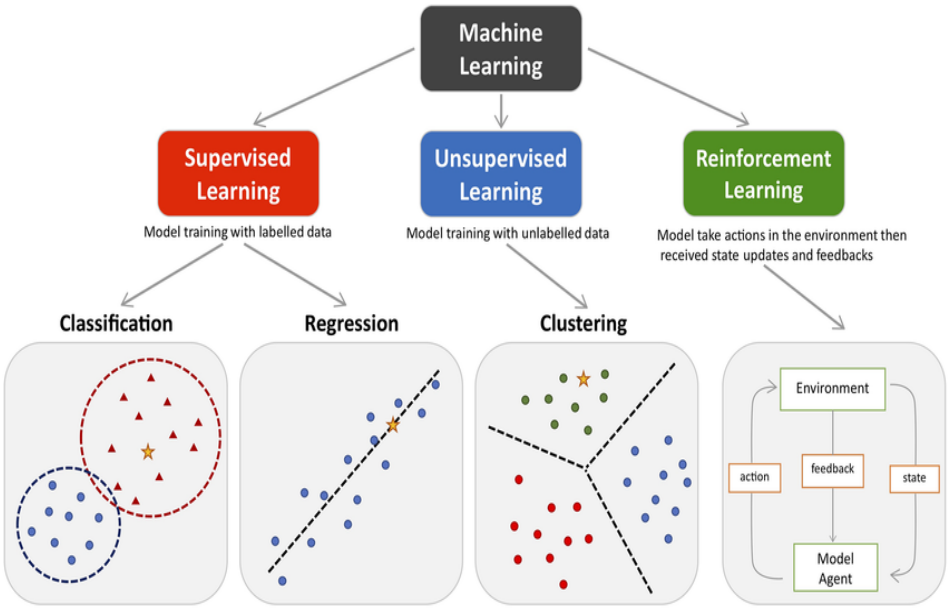

Figure 2.2: *Categories of ML. Figure from [26]*

This study uses supervised learning algorithms to create models that predict a function's outputs based on its corresponding inputs. A fundamental term must be defined before delving deeper into the theory of supervised ML models that were assessed.

A **hyperparameter** is a parameter whose value is set before the learning process begins and controls the behavior and performance of an algorithm, model, or system. Unlike model parameters, which are learned from the data during training, hyperparameters are specified by the user and remain fixed during the learning process. Most hyperparameters involved in this thesis are mentioned in the following section.

2.3 Supervised ML Models for Regression

First, the importance of DB management has to be highlighted. A common approach, mostly in deep learning, is to split the DB into training and validation data. The training DB is used to train the model, and the predictions on the validation DB indicate how well the model can generalize unseen data. A typical data-split is 80% for training and 20% for validation. To achieve maximum generalization capability, overcoming two issues during the learning phase is essential. The first one, called **underfitting**, is when the model hasn't extracted sufficient information from the training DB to map the function of interest. In other words, a model that underfits can be considered undertrained, and it is simpler to think that an undertrained model will perform poorly on both training and unseen data. On the other hand, there is the exact opposite and more tricky situation, known as **overfitting**. In this scenario, the model is overtrained. Having learned the training DB almost perfectly, a model that overfits has also learned complex random patterns present in training DB that do not represent the mapping function of interest at all (noise). Hence, the model that overfits will perform nearly perfectly on the training DB but poorly on unseen data. Therefore, both situations lead to insufficient generalization capability. The challenge is to find the golden mean where the model fits best. Namely, it performs sufficiently on both training and unseen data. While overfitting and underfitting are common issues of supervised ML algorithms, their handling differs from algorithm to algorithm, as each model type has unique characteristics and techniques that are most effective in mitigating them.

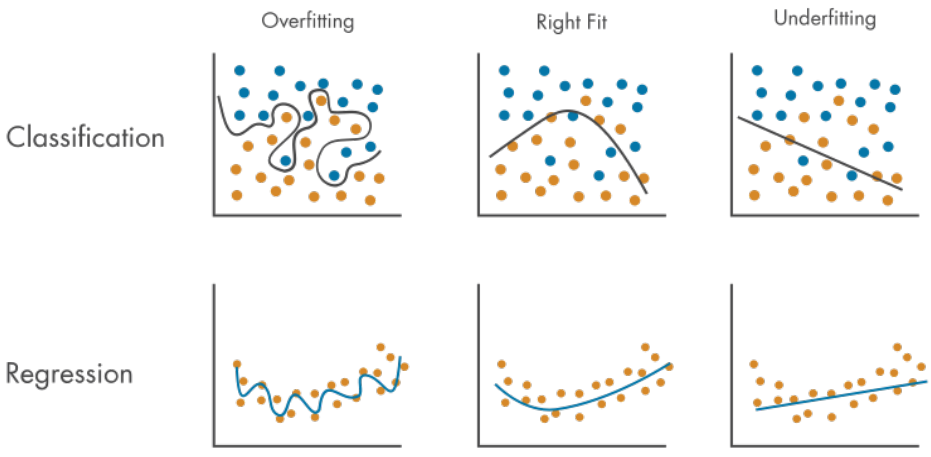

Figure 2.3: *Overfitting, right fit and underfitting. Figure adapted from [27]*

2.3.1 Linear Regression

Linear Regression (LR) is a fundamental supervised learning algorithm used to predict a continuous outcome. In order to map the relationship between the target and the features, it fits a linear equation to the training data. The objective of LR is to find the best-fitting line (in the case of a single input feature) or hyperplane (for multiple input features) that minimizes the error between the predicted and the actual values. The latter is commonly achieved using the ordinary least squares method [28].

2.3.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a simple distance-based supervised learning algorithm for classification and regression. It identifies the K closest data points (neighbors) in the training dataset to a given input based on a specific distance metric (commonly Euclidean distance). For regression tasks, the predictions are the averages of the values of the K nearest neighbors. The only tuning parameter is the value of K, which, as it increases, makes the model more prone to underfitting, while decreasing K makes it prone to overfitting.

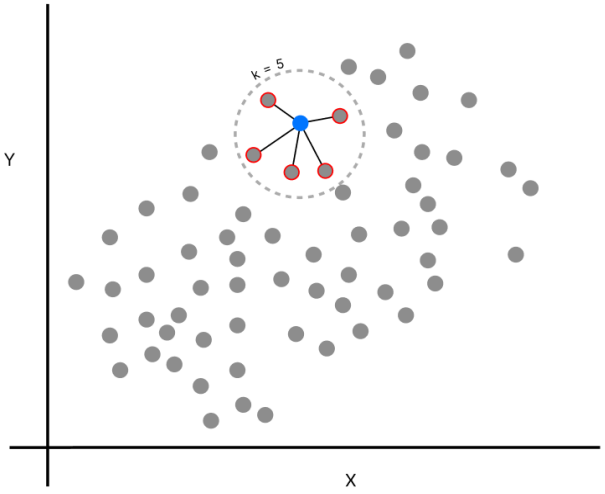

Figure 2.4: *KNN. Figure from [29]*

2.3.3 Support Vector Regression

Support Vector Regression (SVR), is another supervised learning ML algorithm utilized for regression tasks, and it is an extension of Support Vector Machines, which are algorithms used for classification. Thus, its goal is to approximate the relationship between the input features and a continuous target variable, and it does

so by finding a hyperplane that best fits the data points. To do so, it minimizes the prediction error while allowing a certain tolerance (epsilon margin) around the hyperplane. This is achieved by mapping the input variables to a higher-dimensional feature space and finding the hyperplane that minimizes the prediction error while implicitly maximizing the number of data points inside the epsilon margin. Hence, the training of SVR can be viewed as a double-objective optimization: balancing model complexity and prediction error. The points inside the epsilon margin do not contribute to the model's prediction error. SVR has been widely used in CFD applications such as [30, 31]

Regarding SVR's hyperparameters, only three of them, thought to be the most important ones, are being explained. The first two: **epsilon (ε)** and **regularization parameter (C)** are the ones controlling the margin of tolerance. Epsilon (ε) determines the margin of tolerance, while C controls the trade-off between minimizing prediction error and keeping the model simple to generalize better. The third one is the **kernel function** choice. To manage non-linear relationships, SVR applies the kernel function mapping the data into a higher-dimensional space and making linear separation feasible. Commonly utilized kernel functions include linear, polynomial, radial basis functions (RBF), and sigmoid.

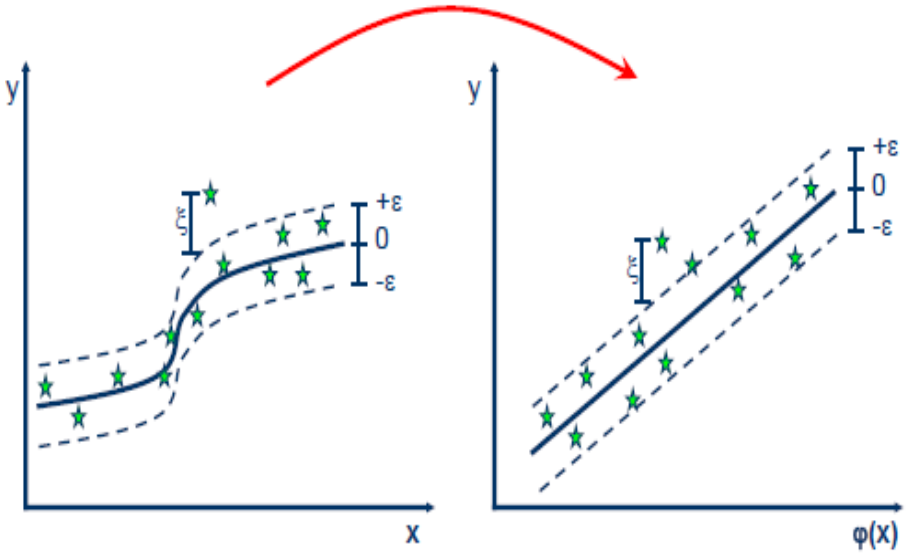

Figure 2.5: *SVR transforms non-linear data from the original input space (left) into a higher-dimensional feature space (right) by applying a kernel function, enabling the model to fit a linear hyperplane within the epsilon margin that captures the underlying data structure. The slack variable ξ_i represents the prediction error for each data point that falls outside the epsilon margin. Figure from [32]*

2.3.4 Deep Neural Networks

Artificial Neural Networks (ANNs) are computing systems designed to mimic the human brain's function and are considered a part of the wider field of machine learning. They are precisely on the dividing line between ML and DL. This is because, in simple terms, Deep Neural Networks (DNNs) are bigger ANNs. They can perform classification and regression but, as mentioned above, this study focuses solely on regression.

The fundamental building block of Deep Learning and Neural Networks is a simple processing unit called perceptron or neuron. Its analogy in the real world is a biological neuron in the human brain. The biological neuron is constructed by three main parts: the Dendrites responsible for receiving information from other neurons; the Soma, which processes this information and produces an output; and the axon, which transmits the output toward other neurons. The biological-artificial neuron analogy is illustrated in figure 2.6. Just like in the brain where billions of these small processing units are interacting and working together, the concept of ANNs is more or less the same. Neurons, that are stacked in layers communicate and exchange information in non-linear ways.

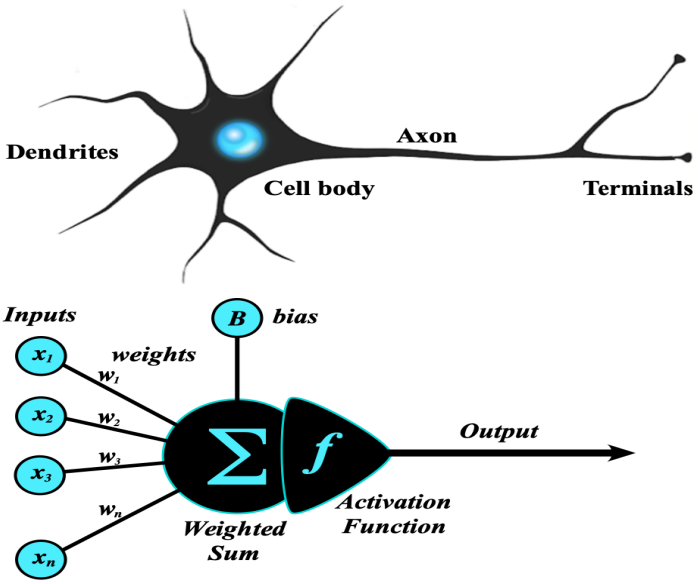

Figure 2.6: *Artificial Neuron: Forward Propagation. Figure from [33]*

Regarding the artificial neuron, the **input vector** $x \in R^n$ is first multiplied by a **weight vector** $w \in R^n$. The latter, contains learnable parameters, the weights, which the network adjusts accordingly to the problem at hand during the training process. In particular, each connection has a corresponding weight parameter the value of which, after the training has ended, represents the strength of the connection. Next, the yielding product $\sum x_i w_i$ is added to the **bias** $b \in R$, the last

learnable parameter of each perceptron. Finally, the resulting linear combination $\sum x_i w_i + b$ is passed through the so-called **activation function (f)**, whose role is to introduce nonlinearity into the network. Each **neuron's output (y)** is a scalar and reads:

$$y = f(\sum x_i w_i + b) \tag{2.1}$$

For the k-est neuron of a particular layer eq. 2.1 reads:

$$y_k = f(\sum x_{ki} w_{ki} + b_k) \tag{2.2}$$

In figure 2.7, one can see artificial neurons stacked in layers and their interaction. The type of network illustrated is known as feedforward fully connected neural network. Note that each layer consists of neurons that do not communicate. In this type of network, each neuron of a hidden layer is connected to all neurons of the previous layer as well as with the ones of the subsequent layer. This way, information taken as input is propagated forward towards the exit of the network (output layer). Conventionally, every layer besides the **input** and the **output layers** is known as **hidden layer**. The distinction between simple (swallow) ANNs and DNNs is often vague and related to the number of hidden layers. In this work, models with more than one hidden layer are considered as DNNs, while the ones with one hidden layer as swallow ANNs. Last, the number of nodes in the input and output layers is equivalent to the mapping function's input and output features (dimensions).

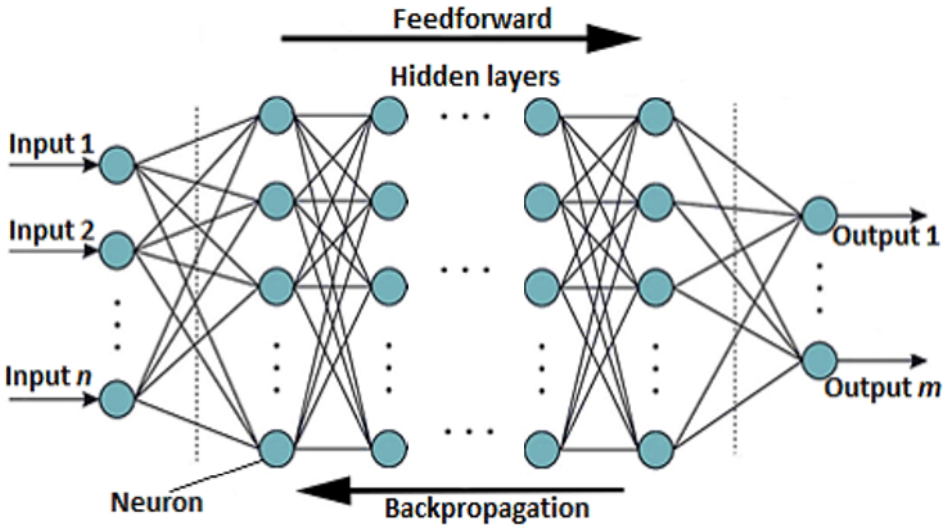

Figure 2.7: *Feedforward fully connected DNN[34]*

A neural network, with a sufficient number of neurons and appropriate activation functions, can approximate any function [35]. The most important user-defined parameters (hyperparameters) are presented along with brief explanations in the following lines.

Hyperparameters

Model Architecture Hyperparameters

1. **Number of layers:** Defines the depth of the network. Additionally, in the context of the aforementioned type of ANN, all fully connected hidden layers are known as dense layers. (In deep learning, there are numerous hidden layer types such as convolutional, pooling, and dropout, among others.)

2. **Number of neurons per layer:** Defines the number of neurons in each hidden layer. As this number increases, the model parameters increase too and the neural network becomes capable to map more complex functions.

3. **Activation functions:** They introduce nonlinearity into the model. Only hidden and output layer neurons possess activation functions. The input layer neurons do not, as they are just responsible for receiving the input features. There are several types of activation functions, some of them presented in figure 2.8.

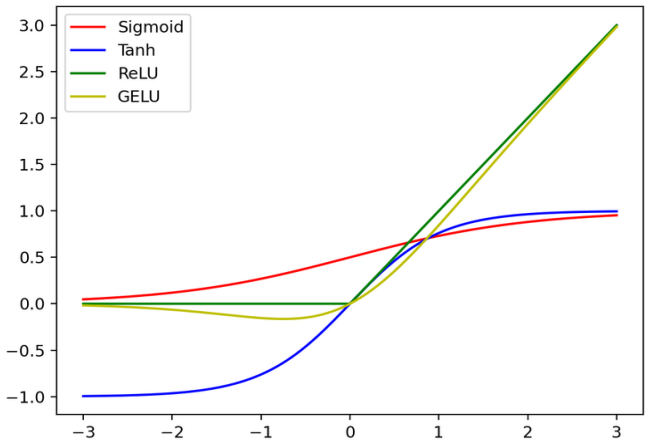

Figure 2.8: *Four common activation functions. Figure from [36]*

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

- **Tanh:**

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

- **ReLU:**

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.5}$$

- **GELU:**

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \tag{2.6}$$

  where $\Phi(x)$ denotes the cumulative distribution function of the Gaussian distribution and erf the error function.

4. **Initialization Methods:** Methods to initialize the weights and biases of the network before the training process. The Xavier initialization method [37] was utilized in this thesis as this is the default in the Tensorflow library.

Training Hyperparameters

When the training of a neural network begins, after the network is initialized, it is provided some input data and produces some outputs that are commonly very far from the target outputs. Thus, an error metric representing the model can be calculated, knowing the target outputs and the model's predictions. The learning phase is a gradient-based optimization problem in which the objective function (to be minimized) is the model's error. At the same time, the design variables are all the weights and biases (a.k.a, all learnable parameters). In every optimization cycle, the model produces a new error value. Next, the gradients of the objective function concerning the design variables (weights and biases) are calculated through a process named **Back Error Propagation** or just **Backpropagation** (presented in appendix A.2). Last, an optimization algorithm exploits the calculated gradients and iteratively updates the design variables, minimizing the model's error solely regarding the training DB. In the following lines, the most crucial training hyperparameters are presented.

5. **Loss Function:** Namely, the training's objective function, also known as **cost function** is the average of the errors of each output neuron. Assuming y_t the target values and y_p the model's predictions, the loss function can be thought of as L(y_t,y_p) and provides an understanding of the distance between the predictions and target outputs. It can be one out of many well-known mean error metrics, yet in this thesis two loss functions, commonly used for regression tasks, were employed: Mean Absolute Error (MAE) and Mean Squared Error (MSE). The last two, along with a third error metric, the Mean Absolute Percentage Error (MAPE), which was only used as a final quality metric for trained models later on, are presented.

   - **Mean Absolute Error (MAE):**

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{ti} - y_{pi}| \tag{2.7}$$

- **Mean Squared Error (MSE):**

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_{ti} - y_{pi})^2 \tag{2.8}$$

- **Mean Absolute Percentage Error (MAPE):**

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_{ti} - y_{pi}}{y_{ti}}| \tag{2.9}$$

where N is the number of outputs.

6. **Learning Rate:** The learning rate, denoted as η, is the user-defined parameter illustrated in eq. 1.1. A relatively small η typically leads to slower but smoother convergence, yet with the risk of becoming trapped in a local extremum. Conversely, a larger η enables quicker progress by taking larger steps towards the optimal, reducing the risk of getting stuck in local extrema. However, as η increases, there is a trade-off: beyond a certain point, the optimization may fail to converge at all, due to the excessively large steps, causing the algorithm to overshoot the optimal point.

7. **Optimizer:** An algorithm responsible for the design variable updates during training must be selected. There are numerous optimizers [38]. In this work, Adaptive Moment Estimation (Adam) was utilized. Adam was created by the combination of two older optimizers (Momentum and RMSprop). Given an initial learning rate value, Adam can adjust its learning rate during optimization.

8. **Validation Split:** This is a value between 0 and 1 which determines what percentages of the available data will be used for training and validation.

9. **Number of Epochs:** An epoch refers to one complete pass of the entire training dataset through the model. The number of epochs indicates how many times the model has processed all the data, and it is used as a termination criterion for training. Setting the number of epochs is crucial because it impacts the yielding model. If the number of epochs is too low, the trained model might not have captured the key patterns in the data, leading to underfitting. On the other hand, if the number is too high, the model will probably learn the noise in the training data, resulting in overfitting and poor generalization capability.

10. **Batch Size:** This hyperparameter represents the number of samples used before each update of the model parameters and can influence whether the optimization algorithm behaves more stochastically or deterministically as the

batch size decreases or increases, respectively. In this work, most models appeared insensitive to changes in batch size, likely due to the small size of the training dataset. As a result, the batch size was always set equal to the size of the training DB.

2.3.5 Learning Curves and Model Checkpoint

The average error computed from predictions on the training DB is known as **train loss**. Additionally, validation data is used to evaluate the model's performance on unseen data during training, providing a new metric known as **validation loss**, which indicates the model's generalization capability. Validation loss is inspected by the user during the training process, usually providing essential information on whether the training must end. The latter is more straightforward to understand by assessing the learning curves (figure 2.9). These line plots are simply the evolution of train and validation losses as the number of epochs increases. One may easily see that initially, both curves drop together. At some point, if the number of epochs is large enough, the validation loss starts to increase, indicating a generalization capability that gets worse after each epoch. The latter is a sign that the model has begun overfitting, and the general rule is that the best-fit model is obtained right before the validation loss starts increasing. Hence, it is assumed that the user should obtain the model created on the epoch with the lowest validation loss. This can be done by stopping the training at that particular epoch, being left with the desired model, or, more conveniently, by using the Model Checkpoint Callback to do something equivalent automatically. Model Checkpoint is a Python tool, configured before the training begins, to monitor the training process, and to save models with the desired characteristics. For example, the Model Checkpoint was used throughout this work to save the model with the lowest validation loss that appeared during training.

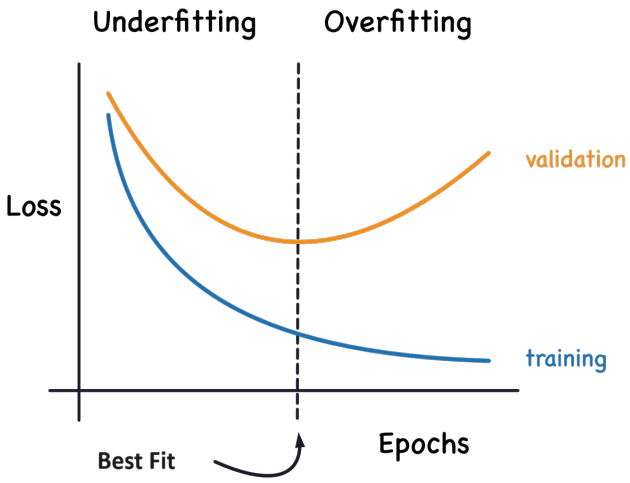

Figure 2.9: *Learning curves. As train and validation losses evolve during training, underfitting and overfitting regions are evident, and one should obtain a model as close to the best-fit line as possible. Figure adapted from [27]*

Finally, it must be noted that many ML algorithms especially DNNs must be provided with non-dimensionalized data. The two most known non-dimensionalization techniques are presented in appendix A.1.2.

2.4 Ensemble Learning

Ensemble learning combines two or more individual learners aiming to improve predicting performance. The most widely used techniques are Bagging, Boosting, and Stacking. While Bagging and Boosting are homogenous ensembles, meaning all combined models must be of the same type, with Stacking, this is not the case. Stacking is a heterogeneous ensemble method, making it possible to combine completely different base learners.

- **Bagging** or **Bootstrap Aggregating** is an ensemble method that combines a model's multiple instances, trained, in parallel, on different subsets of the training data through bootstrapping (randomly selecting data and allowing for duplicates). The base learners' prediction is combined by averaging (for regression) or voting (for classification).
- **Boosting** is a sequential ensemble technique where each model is trained to correct the predictions of the previous one.
- **Stacking** or **Stacked Generalization** is the ensemble method employed in this work to combine two models of different types. Instead of simply averaging or voting, stacking involves training a meta-learner who receives the

predictions of the base learners as inputs and is responsible for best combining them to make the final prediction. Stacking flowchart is illustrated in figure 2.10.

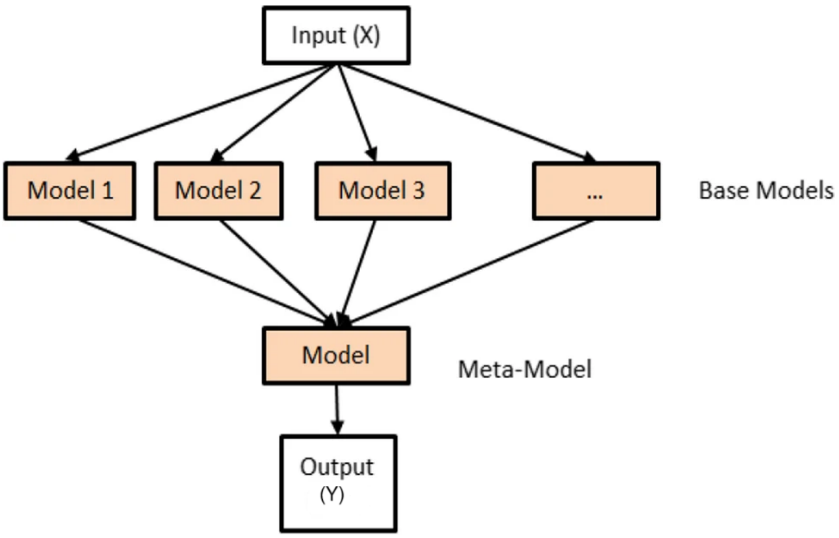

Figure 2.10: *Stacking ensemble method. Multiple models of different types provide their predictions as training data to the meta-learner, which is responsible for generating the final predictions. Figure adapted from [39].*

Chapter 3

The CFD Solver

3.1 Mean Flow Equations

The CFD tool utilized in this work is the GPU-accelerated code PUMA, developed by PcOpt/NTUA [40]. It solves the Reynolds Averaged Navier-Stokes (RANS) equations for steady flows of compressible fluids. The mean-flow equations are:

$$\frac{\partial U_n}{\partial \tau} + \frac{\partial f_{nk}^{\text{inv}}}{\partial x_k} - \frac{\partial f_{nk}^{\text{vis}}}{\partial x_k} = 0 \tag{3.1}$$

and 3.1 is solved for the conservative flow variables $\mathbf{U} = [\rho \quad \rho v_1 \quad \rho v_2 \quad \rho v_3 \quad \rho E]^T$, where ϱ stands for the fluid's density, v_k are the Cartesian velocity components, E the total energy per unit mass and p the pressure. τ is the pseudo-time step and x_k the Cartesian coordinates. f_{nk}^{inv} and f_{nk}^{vis} are the inviscid and the viscous fluxes, respectively. The term f_{nk}^{vis} involves the stresses which are given by the following formula:

$$\tau_{km} = (\mu + \mu_t)\left(\frac{\partial v_k}{\partial x_m} + \frac{\partial v_m}{\partial x_k} - \frac{2}{3}\delta_{km}\frac{\partial v_e}{\partial x_e}\right) \tag{3.2}$$

where μ and μ_t represent the dynamic and turbulent viscosity. δ_{km} is the Kronecker delta.

3.2 Turbulence Model

The turbulence is modeled using the one equation Spalart-Allmaras model. It solves the equation 3.3 presented swiftly for the sake of completeness. For more information about the equation terms, one must see [41].

$$\frac{\partial(\rho\tilde{\nu})}{\partial\tau} + \frac{\partial(\rho\tilde{\nu}v_k)}{\partial x_k} - \frac{\partial}{\partial x_k}\left[\rho\left(\nu+\tilde{\nu}\right)\frac{\partial\tilde{\nu}}{\partial x_k} + c_{b2}\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k}\right] - \tilde{P}_{\tilde{\nu}} + D_{\tilde{\nu}} = 0 \tag{3.3}$$

It is solved by computing the $\tilde{\nu}$, which is then used to obtain the turbulent viscosity μ_t using

$$\mu_t = \rho\tilde{\nu}f_{v1} \tag{3.4}$$

where the term f_{v1} can also be found in [41].

3.3 Transition Model

The transition point, where the laminar flow becomes turbulent and vice-versa, is critical in ASO. The model used to simulate this phenomenon is the $\gamma - \tilde{R}e_\theta$ transitional model [42], which solves the following two equations: the one of intermittency γ, and the one of the transition momentum thickness Reynolds number $\tilde{R}e_\theta$.

$$\frac{\partial(\rho\gamma)}{\partial\tau} + \frac{\partial(\rho v_k\gamma)}{\partial x_k} - \frac{\partial}{\partial x_k}\left[\left(\mu+\frac{\mu_t}{\sigma_f}\right)\frac{\partial\gamma}{\partial x_k}\right] - P_\gamma + D_\gamma = 0 \tag{3.5}$$

$$\frac{\partial(\rho\tilde{R}_{e_{\theta t}})}{\partial\tau} + \frac{\partial(\rho v_k\tilde{R}_{e_{\theta t}})}{\partial x_k} - \frac{\partial}{\partial x_k}\left(\sigma_{\theta,t}(\mu+\mu_t)\frac{\partial\tilde{R}_{e_{\theta t}}}{\partial x_k}\right) - P_{\theta,t} - D_{SCF} = 0 \tag{3.6}$$

where the production and destruction terms are the following:

$$P_\gamma = \rho c_{\alpha_1}F_{\text{length}}F_{\text{onset}}\left[\phi_{-300}\left(\zeta, \frac{M\sqrt{MRe}}{20}\right)\right]\sqrt{\gamma}(1-c_{\epsilon_1}\gamma) \tag{3.7}$$

$$D_\gamma = \rho c_{\alpha_2}F_{\text{turb}}\left[\phi_{-300}\left(\zeta, \frac{M\sqrt{MRe}}{20}\right)\right]\gamma(c_{\epsilon_2}\gamma-1) \tag{3.8}$$

$$P_{\theta,t} = \rho \frac{c_{\theta,t}}{T} \left(Re_{\theta t}^{\mathrm{eq}} - \tilde{Re}_{\theta t} \right) (1 - F_{\theta t}) \tag{3.9}$$

$$D_{SCF} = c_{\theta,t} \frac{\rho}{T} c_{\mathrm{crossflow}} \min \left(Re_{\mathrm{SCF}} - \tilde{Re}_{\theta t}, 0 \right) F_{\theta t} \tag{3.10}$$

$$Re_{SCF} = -35.088 \ln \left(\frac{h_{rms}}{\theta_t} \right) + 319.51 + f(DH_{CF+}) - f(DH_{CF-}) \tag{3.11}$$

The model closure constants are $c_{a1} = 2$, $c_{a2} = 0.06$, $c_{\varepsilon 1} = 1$, $c_{\varepsilon 2} = 50$, $c_{\theta,t} = 0.03$, $c_{crossflow} = 0.6$, $\sigma_{\theta,t} = 2$, $\sigma_f = 1$. The γ field, after being computed, is being used inside the Spalart-Allmaras model equation, and, in particular, by affecting the production term $\tilde{P}_{\tilde{\nu}}$ as shown in

$$\tilde{P}_{\tilde{\nu}} = \gamma \rho c_{b1} \tilde{S} \tilde{\nu}. \tag{3.12}$$

3.4 The flow solver - PUMA

In this Diploma Thesis, the flow field around an airfoil and a wing is simulated, as stated above, using the compressible, GPU-accelerated flow solver PUMA. First the eqs. 3.1 3.3 3.5 and 3.6 are discretized using the finite volume method, and then, the discretized equations are solved using a pseudo-time marching algorithm, specifically a multi-stage Runge-Kutta. The computations are performed on unstructured meshes that consist of elements such as tetrahedra, pyramids, prisms, and hexahedra. The integration is carried out over vertex-centered finite volumes. Figure 3.1 is a flowchart respresenting the order that PUMA solves the primal equations on each iteration.

Figure 3.1: *Solution of the Primal equations in every iteration. The 5 Mean Flow equations are solved producing 5 flow quantities. Next, after Spallart-Allmaras is solved for $\rho\tilde{\nu}$, the new value for μ_t is obtrained. Last, the two equations of $\gamma - \tilde{Re}_\theta$ are solved for $\rho\gamma$ and $\rho\tilde{Re}_\theta$.*

Chapter 4

UQ Methods

4.1 Introduction to UQ

UQ is a field that utilizes math, statistics, and probability theory to estimate, propagate, and limit uncertainty in system representations, a.k.a. models [43]. Mathematical models in engineering, healthcare, economics, and many other fields provide the fundamental information that leads to decision-making [44]. The latter highlights the importance and need for model reliability and, e.g., in ASO, solution robustness [45]. Uncertainty can be divided into two types: **aleatoric** and **epistemic**. The word "aleator" in Latin refers to someone who rolls the dice, and thus, aleatoric uncertainty is related to the inherently random nature of the system under study. The word "ἐπιστήμη" from Greek means knowledge. Hence, epistemic uncertainty is associated with a lack of knowledge. Aleatoric uncertainty is a type of uncertainty that cannot be minimized but can be recognized and measured. By contrast, epistemic uncertainty can be reduced by improving the data or the information about the system [46]. UQ, while the heart of RDO, usually increases the computational cost by orders of magnitude, especially when the optimization is to be carried out with evolutionary algorithms. This thesis strives to develop surrogate models to be used as low-cost evaluation tools in quantifying the end-to-end propagation of uncertainty (UQ) within an objective function F (the output of the CFD model). To do so, it is necessary to quantify the model's output fluctuations, as noted in chapter 1, usually computing their first two statistical moments: the mean (μ) and the variance (σ^2). The two methods employed for the task of UQ are described in the following.

4.2 Monte Carlo Simulation

The Monte Carlo method is a computational technique used to estimate an uncertain event's possible outcomes or approximate solutions to mathematical problems that include random variables. It's named after the famous gambling location in Monaco and was invented in the late 1940s by Stanislaw Ulam while he was working on nuclear weapon projects at the Los Alamos National Laboratory [47, 48]. First, the predictive model is defined along with its inputs and outputs. Next, the user has to model the uncertain input variables, each with a probability distribution that describes it better. Additionally, random sampling is applied to the input variable space, and random samples of every uncertain variable w.r.t. its corresponding probability distribution are obtained. Finally, the model evaluates the samples, to compute the outputs for each sample; upon these, statistical conclusions can be drawn. Due to the law of large numbers, it is evident that the bigger the sample size, the more accurate the results of the Monte Carlo. However, the large number of model evaluations demands a lot of computational power and often makes the method prohibitively expensive. The problems studied in this thesis, in which computationally expensive CFD codes hold the role of the high-fidelity model, fall under the last category, and so the MC method is mainly used with surrogate models.

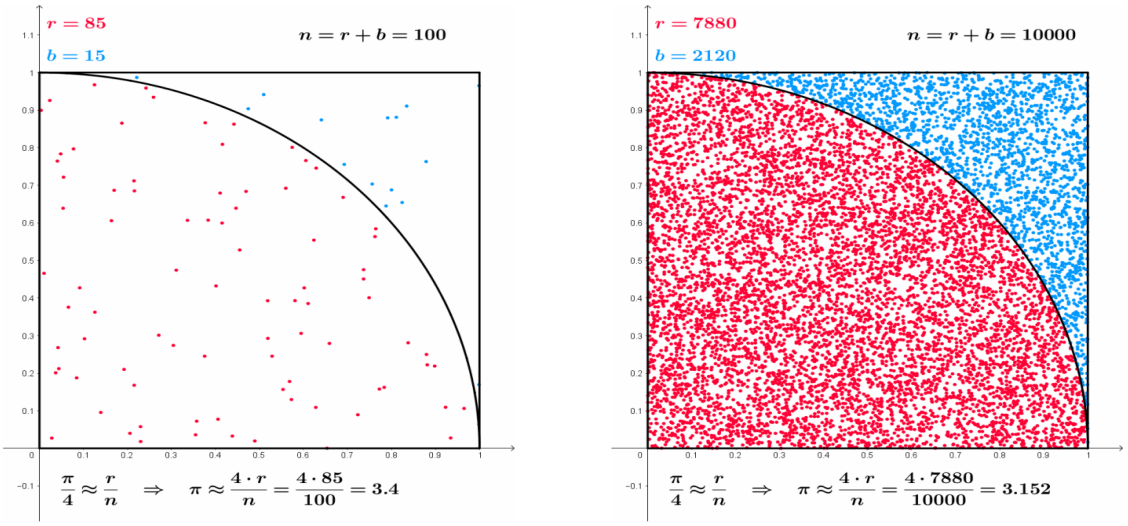

Figure 4.1: *Monte Carlo Simulation example. Early stage of the simulation (left), final stage of the simulation (right). The goal is to calculate the value of π. The square's side length is 1 for simplicity. The area of the quarter circle is* $\frac{\pi}{4} = \frac{\pi}{4} \cdot 1^2$. *Points are uniformly distributed within the square, and the model checks if each point falls inside the quarter circle. Let r be the number of points inside the quarter circle and n be the total number of points. The fraction* $\frac{r}{n}$ *approximates the actual area ratio* $\frac{\frac{\pi}{4}}{1}$ *as n increases. Figure adapted from [49]*

In this work, the use of MC for UQ, assuming a problem of M uncertain variables,

goes as follows. Initially, a number of random vectors (N) must be drawn from their joint distribution $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_M$. To do so, N random numbers are sampled from each distribution $\mathcal{D}_i$, leading to the creation of the following uncertain sample vectors:

$$\begin{aligned} \vec{x}^{(1)} &= \begin{pmatrix} x_1^{(1)} & \cdots & x_M^{(1)} \end{pmatrix}, \\ \vec{x}^{(2)} &= \begin{pmatrix} x_1^{(2)} & \cdots & x_M^{(2)} \end{pmatrix}, \\ &\vdots \\ \vec{x}^{(N)} &= \begin{pmatrix} x_1^{(N)} & \cdots & x_M^{(N)} \end{pmatrix} \end{aligned}$$

Subsequently, some model, e.g., the RANS solver, has to compute their corresponding responses (QoIs):

$$\vec{y}^{(j)} = F(\vec{x}^{(j)}) \quad , \quad j \in 1, \ldots, N$$

Hence, a DB is created upon which the well-known statistical formulas discussed in A.1.1 are assessed to obtain the mean and standard deviation of each QoI.

4.3 Non-Intrusive Polynomial Chaos Expansion

PCE originates from Norbert Wiener's publication "The Homogeneous Chaos" [50] in 1938. It is about the orthogonal decomposition of a random variable into a suitable series, aiming to determine analytically the statistical moments of the truncated expansion. The maximum degree of the expansion is known as "chaos order". PCE is divided into two categories: the intrusive (iPCE) and the non-intrusive (niPCE). Their names are related to whether the user needs to modify the primal solver or not. The iPCE generally performs better in terms of cost, yet its use is more complex, and one must have access to the source code. This thesis focuses only on niPCE; in particular, two variants of the latter are utilized for UQ. In the niPCE, the objective function (QoI) is handled as a "black box" and expanded as a linear combination of a family of orthogonal polynomials. At first, PCE could only work with random variables that followed the Gaussian distribution, employing the Hermite polynomials, but nowadays, it can be applied to uncertain variables of any distribution. This generalization of PCE states that for each Probability Density Function (PDF) used to model the uncertain variables, there is a specific corresponding family of orthogonal polynomials appropriate for expanding the QoI. However, throughout the present work, they are all assumed to be normally distributed by assessing the Three Sigma rule for modeling the uncertain variables. The two niPCE variants (that are being explained later on in the present chapter) are related to the mathematical approach of some calculations required by the method and are known as Gauss Quadrature niPCE (gPCE) and Regression assisted niPCE (rPCE).

Considering a problem with only one uncertain variable $x \in \mathbb{R}$ which follows a PDF defined as w(x) and $\mathcal{P} = \{p_0(x), p_1(x), \dots, p_k(x), \dots\}$ a family of polynomials p_i, with i being the maximum rank of each polynomial and k the chaos order. In accordance with the PC theory, $F(x)$ can be approximated by a different function $f(x)$ with the same stochastic input x, defined as a linear combination of the polynomials belonging in $\mathcal{P}$:

$$F(x) \approx f(x) = \sum_{i=0}^{\infty} a_i p_i(x) \tag{4.1}$$

where $a_i \in \mathbb{R}$ and $f : \mathbb{R} \to Y \subseteq \mathbb{R}$. As things stand, the cut-off point of the expansion or chaos order (k) must be selected. It's obvious that k represents the accuracy level of the truncated expansion, and a bigger k corresponds to better accuracy yet with increased computational cost.

The n-th statistical moment of the set Y can be computed as:

$$\langle y^n \rangle = \int_D (f(x))^n w(x) dx = \int_D \left(\sum_{i=0}^{k} a_i p_i(x) \right)^n w(x) dx$$

$$\Rightarrow \langle y^n \rangle = \int_D \left(\sum_{i_1=0}^{k} a_{i_1} p_{i_1}(x) \right) \cdots \left(\sum_{i_n=0}^{k} a_{i_n} p_{i_n}(x) \right) w(x)\, dx$$

$$\Rightarrow \langle y^n \rangle = \sum_{i_1=0}^{k} \cdots \sum_{i_n=0}^{k} a_{i_1} \dots a_{i_n} \int_D p_{i_1}(x) \dots p_{i_n}(x) w(x) dx \tag{4.2}$$

Due to its polynomial nature, the integral in eq. 4.2 can be solved analytically. Thus, any statistical moment of the function F can be computed. Moreover, the equation becomes even simpler by making $\mathcal{P}$ a family of orthogonal polynomials.

4.3.1 Orthogonal Polynomials

As mentioned above, the most important realization is that for every PDF, w(x), there is a corresponding family $\mathcal{P}$ of orthogonal polynomials. This type of polynomials is characterized by the property that their inner product (Galerkin projection) w.r.t w(x) reads:

$$\langle p_i(x), p_j(x) \rangle_w = \int_D p_i(x) p_j(x) w(x)\, dx = \langle p_i(x), p_i(x) \rangle_w \delta_i^j \tag{4.3}$$

where δ_i^j is the Kronecker delta:

$$\delta_i^j = \begin{cases} 0, & \text{for} \quad i \neq j \\ 1, & \text{for} \quad i = j \end{cases}$$

For i = j, eq.4.3 becomes:

$$\langle p_i(x), p_j(x) \rangle_w = \int_D P_i^2(x) w(x)\, dx = \|p_i\|_w^2 = \gamma_i \tag{4.4}$$

In this case, the Galerkin projection is equal to the w-norm, and the normalization metric $\sqrt{\gamma_i}$ of the p_i polynomial is defined. The $\sqrt{\gamma_i}$ is named after the fact that it's utilized to normalize the polynomials by creating the following condition.

When $\|p_i\|_w^2 = 1$, then, the orthogonal polynomials are also normalized and so, are now called "orthonormal".

It is highlighted once more that each stochastic distribution corresponds to a specific family of orthogonal polynomials in a particular domain and with a specific PDF, w(x). The most common ones are in table 4.1.

Distribution	**Probability Density**	**Orthogonal Polynomials**	**Support Range**
Normal	$\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$	Hermite $H_n(x)$	$[-\infty, +\infty]$
Uniform	$\frac{1}{2}$	Legendre $P_n(x)$	$[-1, +1]$
Beta	$\frac{(1-x)^\alpha (1+x)^\beta}{2^{\alpha+\beta+1} B(\alpha+1, \beta+1)}$	Jacobi $P_n^{(\alpha,\beta)}(x)$	$[-1, +1]$
Exponential	e^{-x}	Laguerre $L_n(x)$	$[0, +\infty)$
Gamma	$\frac{x^k e^{-x}}{\Gamma(k+1)}$	Generalized Laguerre $L_n^{(k)}(x)$	$[0, +\infty)$

Table 4.1: *Distributions, their densities, orthogonal polynomials, and support ranges.*

The last useful feature is that all of the aforementioned polynomial families $\mathcal{P}$ share the following characteristic: their first polynomial of zero degree is always

$$p_0(x) = 1 \tag{4.5}$$

In the case of normally distributed random variables, the Hermite polynomials are utilized, and this scenario is illustrated for a 1D Hermite polynomial. The weight function (PDF) of a single random variable that follows the Gaussian distribution is given by:

$$N(\mu, \sigma) : w(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{4.6}$$

where μ and σ are the variable's mean and standard deviation, respectively. The "probabilistic" Hermite polynomials [5] are presented:

$$\begin{aligned}
H_0(x) &= 1, \\
H_1(x) &= x, \\
H_2(x) &= x^2 - 1, \\
H_3(x) &= x^3 - 3x, \\
H_4(x) &= x^4 - 6x^2 + 3, \\
H_5(x) &= x^5 - 10x^3 + 15x, \\
&\vdots
\end{aligned}$$

The recurrence relation is given by:

$$H_{k+1}(x) = xH_k(x) - kH_{k-1}(x) \tag{4.7}$$

Their w-norm is:

$$\|H_k\|_w^2 = \gamma_k = \langle H_k(x), H_k(x) \rangle_w = \int_{-\infty}^{\infty} H_k^2(x) w(x)\, dx = k! \tag{4.8}$$

Finally, this scenario can also be further simplified by dividing every polynomial with the appropriate normality metric, obtaining the orthonormal Hermite polynomials:

$$\tilde{H}_k(x) := \frac{H_k(x)}{\|H_k\|_w} = \frac{H_k(x)}{\sqrt{k!}} \quad \Rightarrow \quad \|\tilde{H}_k(x)\|^2 = \gamma_k = 1, \quad \forall k = 0, 1, 2, \ldots \tag{4.9}$$

4.3.2 Mean and Variance

Although this thesis does not discuss the categories of statistical moments in detail, it should be clarified that equation 4.2 calculates the m-th raw statistical moment. The first raw statistical moment is the mean. However, the second raw statistical moment is not the variance. Its quantity is used along with the mean to produce

the second central statistical moment, known as variance [51]. This clarification is crucial as the terms "first" and "second" statistical moments are normally used to describe the mean and variance. Thus, the mean and variance of a set Y are connected with the following formula:

$$\sigma_Y^2 = \langle y^2 \rangle - \mu_Y^2 \tag{4.10}$$

where $\langle y^2 \rangle$ is the second raw statistical moment.

The first statistical moment derived from eq. 4.2 is the following:

$$\mu_Y = \langle y \rangle = \int_D (f(x))^1 w(x)\, dx = \int_D f(x) w(x)\, dx$$

$$= \int_D \left(\sum_{i=0}^{k} a_i P_i(x) \right) w(x)\, dx = \sum_{i=0}^{k} a_i \int_D P_i(x) w(x)\, dx \tag{4.11}$$

Therefore, since $\int_D w(x)\, dx = 1$, and $P_0(x) = 1$, there is:

$$\int_D p_i(x) w(x)\, dx = \int_D p_i(x) \cdot 1 \cdot w(x)\, dx = \int_D p_i(x) p_0(x) w(x)\, dx = \delta_i^0$$

and so, equation 4.11 becomes: $\mu_Y = a_0$.

The second statistical moment, derived also from eq.4.2, is:

$$\langle y^2 \rangle = \int_D (f(x))^2 w(x)\, dx = \int_D \left(\sum_{i=0}^{\infty} a_i p_i(x) \right)^2 w(x)\, dx$$

$$= \sum_{i_1=0}^{\infty} \sum_{i_2=0}^{\infty} a_{i_1} a_{i_2} \int_D p_{i_1}(x) p_{i_2}(x) w(x)\, dx$$

which due to the orthogonality property, eq.4.3, becomes:

$$\langle y^2 \rangle = \sum_{i=0}^{\infty} a_i^2 \int_D p_i(x)^2 w(x)\, dx = \sum_{i=0}^{\infty} a_i^2 \| p_i(x) \|_w^2$$

and by exploiting the orthonormal polynomials, eq.4.3.1, this equation is further simplified as:

$$\langle y^2 \rangle = \sum_{i=0}^{\infty} a_i^2 \tag{4.12}$$

The standard deviation is computed by eq. 4.10 as follows:

$$\sigma_Y = \sqrt{\langle y^2 \rangle - \mu_Y^2} = \sqrt{\sum_{i=0}^{\infty} a_i^2 - a_0^2} \Rightarrow \sigma_Y = \sqrt{\sum_{i=1}^{k} a_i^2} \tag{4.13}$$

4.3.3 PCE Coefficients and Generalization to Multiple Dimensions

The first approach in calculating the k+1 coefficients $a_i, i = 0, 1, ..., k$ uses the polynomials' inner product along with the orthogonality and normality properties.

$$\langle f(x), \tilde{p}_i(x) \rangle_w = \int_D f(x) p_i(x) w(x)\, dx \approx \int_D F(x) p_i(x) w(x)\, dx \tag{4.14}$$

$$\langle f(x), \hat{p}_i(x) \rangle_w = \left\langle \sum_{\lambda=0}^{k} a_\lambda \hat{p}_\lambda(x), \hat{p}_i(x) \right\rangle_w = a_i \|\hat{p}_i(x)\|_w^2 = a_i \tag{4.15}$$

and from eqs. 4.14 and 4.15, it is:

$$a_i = \int_D F(x) \hat{p}_i(x) w(x)\, dx \tag{4.16}$$

Hence, by solving the integral, one can obtain the PCE coefficients.

So far, all discussions refer to **1Dimensional** functions approximated by univariate orthonormal polynomials. It is proven [5] that PCE's governing equations generalize to **multiple dimensions** as follows:

Let F be the function that depends on the uncertain variables $c_i \in [1, M]$. F can be approximated by niPCE as:

$$F(\vec{c}) \approx \sum_{i=0}^{Q-1} J_i P_i(\vec{c}) \tag{4.17}$$

where $Q = \frac{(M+k)!}{M!k!}$ is the largest degree of the multivariate orthonormal polynomials

$P_i(\vec{c})$, and J_i are their corresponding weights. The multivariate polynomials $P_i(\vec{c})$ are constructed through the products of univariate orthogonal polynomials of the same family. The mean and standard deviation are computed as follows:

$$\mu_F = J_0, \quad \sigma_F = \sqrt{\sum_{i=1}^{Q-1} J_i^2} \tag{4.18}$$

and eq.4.16 becomes:

$$J_i = \int \cdots \int F(\vec{c}) P_i(\vec{c}) W(\vec{c}) \tag{4.19}$$

where $W(\vec{c})$ is the product of the PDFs $w_i, i \in [1, M]$ of all uncertain variables.

4.3.4 Integration using Gaussian Quadrature

The focus is on how integrals 4.16, 4.19 can be calculated with the less F's evaluations. A reminder that in ASO problems, F refers to the primal problem, and one evaluation of F corresponds to 1 TU. Therefore, the aim is to compute the integral most efficiently. The numerical method employed for this task, taking advantage of the fact that the integral of interest is a weighted polynomial, is the **Gaussian Quadrature (GQ)**. Hence, this first niPCE variant can be found in the name of gPCE throughout this thesis.

In simple terms, the GQ method requires the calculation of F at several z_j points (known as gauss nodes), which are the $p_{k+1}(x)$ polynomial's roots and their population (n) is equal to the degree of that polynomial. Every $F(z_j)$ is multiplied with a corresponding weight ω_j, where both gauss nodes and weights are selected in a way aiming to minimize the GQ method's error, and the sum of all products, represents the integral's result. For example, if the function of interest is g(x)= w(x)f(x), where w(x) is the previous weight function and f(x) is the polynomial approximation of F(x), the method stands as follows:

$$\int_D g(x)\,dx = \int_D w(x) f(x)\,dx = \sum_{j=1}^{n} \omega_j f(z_j) \tag{4.20}$$

Gauss Quadrature is divided into slightly different variations, depending on the polynomial family $P_i(\vec{c})$ utilized to expand F (Hermite, Legendre, Jacobi, etc.). For the currently discussed scenarios concerning only normal distributions, the integration method is known as Gauss-Hermite Quadrature(GHQ). Last, the GQ method is accurate, yet it has a significant disadvantage. In order to compute J_i, eq.4.19, $(k+1)^M$ evaluations of F are required. This exponential relationship between the

cost and the number M of the uncertain variables showcases the so-called "curse of dimensionality" involved in the method. The latter makes the gPCE method prohibitively expensive for problems with even a moderate number of uncertain variables.

4.3.5 PCE Coefficients with Regression Approach

The second way to compute the J_i, namely the second niPCE variant, denoted as rPCE, is a method based on linear regression [52, 53, 54]. It does not require numerical integration, and it practically uses the eq. 7.21 directly to calculate the coefficients. In particular, F is being computed at L different $\vec{c}$ (uncertain variable vectors samples). Having done so, equation 7.21 is utilized to form a system of L equations and Q unknowns.

$$\begin{bmatrix} P_0(c_1) & P_1(c_1) & \cdots & P_Q(c_1) \\ P_0(c_2) & P_1(c_2) & \cdots & P_Q(c_2) \\ \vdots & \vdots & \ddots & \vdots \\ P_0(c_L) & P_1(c_L) & \cdots & P_Q(c_L) \end{bmatrix} \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_Q \end{bmatrix} = \begin{bmatrix} F(c_1) \\ F(c_2) \\ \vdots \\ F(c_L) \end{bmatrix} \tag{4.21}$$

L can not be lower than Q ($Q = \frac{(M+k)!}{M!k!}$). If L=Q, the system can be solved directly. One may observe that rPCE is not subjected to the "curse of dimensionality" like gPCE as the number of F evaluations is not exponentially related to M. Aiming for more accuracy, F is usually oversampled by some oversampling ratio, e.g., L=3Q, and the system becomes overdetermined. Then the system's solution must be obtained with regression, commonly using the least squares method. The arising question is how these collocation points must be selected ? In the present work, rPCE utilizes the output of F evaluated on GQ nodes or on nodes obtained with the Latin Hypercube Sampling (LHS) method.

Chapter 5

UQ in an Airfoil Case

This chapter assesses the use of DNNs for UQ in the case of a flow problem around an isolated airfoil. The DNNs are compared with the CFD and the surrogate RBFN used in [54] for the same purpose. By extending the work of [54], the DNNs are trained and presented next to the RBFNs. Four uncertainties associated with four (M=4) constants $(c_{a1}, c_{a2}, c_{\varepsilon 2}, c_{\theta,t})$ of the $\gamma - \tilde{Re}_\theta$ transition model are considered. The latter were selected as the sources of uncertainty in the model, as they were calibrated based on empirical information and, thus, might not be best-suited for some other cases. The aim is to quantify their impact on the lift and drag coefficients (C_L and C_D) of the airfoil. In specific, all uncertain variables are assumed to be normally distributed around their nominal values (mean values) with arbitrarily chosen (10% of their mean values) standard deviations (figure 5.1).

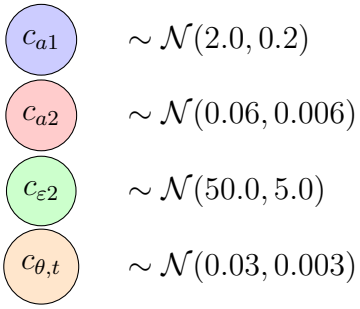

Figure 5.1: *NLF(1)-0416 Airfoil Case. Uncertain variables*

The models, are trained to receive the uncertain variable vectors as inputs, and predict the, otherwise computed by running an expensive CFD code, desired QoIs. Moreover, utilizing the surrogates, a sensitivity analysis is conducted for the three UQ methods under consideration, as this is much cheaper (nearly free with a trained surrogate) than doing this with the CFD code.

5.1 The NLF(1)-0416 Airfoil

5.1.1 Computational Mesh and Aerodynamic Polar Diagram

Initially, a C-type structured mesh of 705x97 nodes, formed by quadrilaterals, (figure 5.2) is used, which was found in [55], a transition modeling workshop. PCOpt's in-house GPU-accelerated PUMA software works only with unstructured mesh, so the aforementioned structured mesh is handled as an unstructured one by the CFD code.

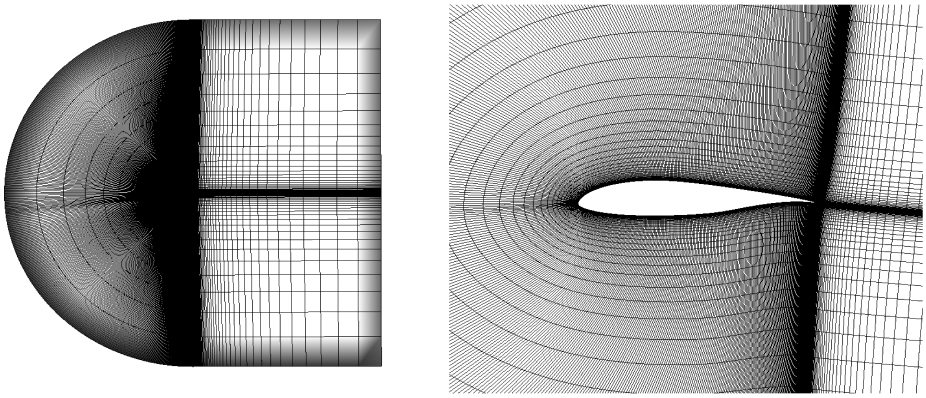

Figure 5.2: *NLF(1)-0416 Airfoil Case. Left: computational mesh around the Airfoil. Right: close-up view of the mesh near the Airfoil.*

The turbulence was modelled with the one equation Spalart-Allmaras turbulence model, and the transition effect with $\gamma - \tilde{Re}_\theta$ transition model. The turbulence viscosity (illustrated, for 4 AoA, in figure 5.5) has very small values in areas with laminar flow and its magnitude gets higher in the turbulent flow part.

To validate the results of the CFD code in the case of a 2D airfoil, one should generate the aerodynamic polar diagram and compare it with experimental data. An aerodynamic polar is a graphical representation that illustrates the aerodynamic characteristics of an airfoil or wing section at various Angles of Attack (a). This plot typically displays the three key aerodynamic coefficients: lift coefficient (C_L), drag coefficient (C_D), and moment coefficient (C_M).
Experimental data for the NLF(1)-0416 at flow conditions $M_\infty = 0.3$ and $Re_c = 6 \cdot 10^6$ can be found in [56]. CFD runs took place for the extracted angles, having the turbulence and transition models enabled, and the C_L-a is plotted and compared with experimental data (figure 5.3).

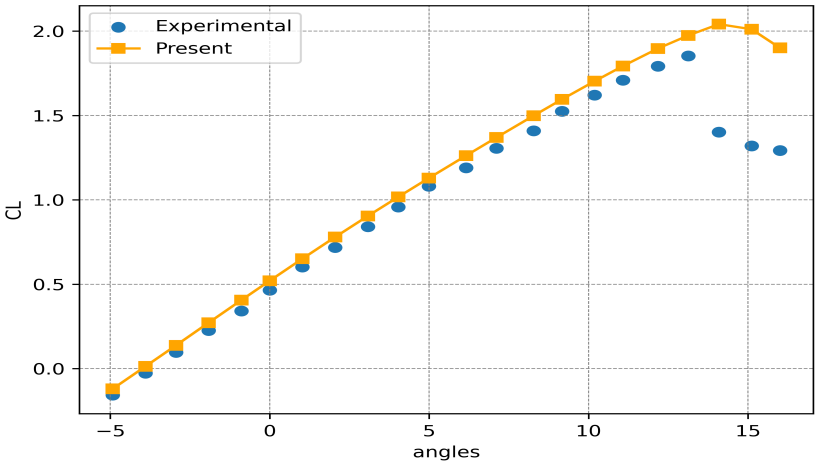

Figure 5.3: *NLF(1)-0416 Airfoil Case. C_L across a range of angles of attack. The present results accurately match the experimental data before the stall effect occurs. At the so-called critical angle of attack, the flow becomes massively separated and, after that, there is a noticeable decrease in C_L. One can observe some differences between experimental and present data in this area.*

As regards the C_D and C_M, they were also computed for the same angles and are presented on top of experimental data (figure 5.4).

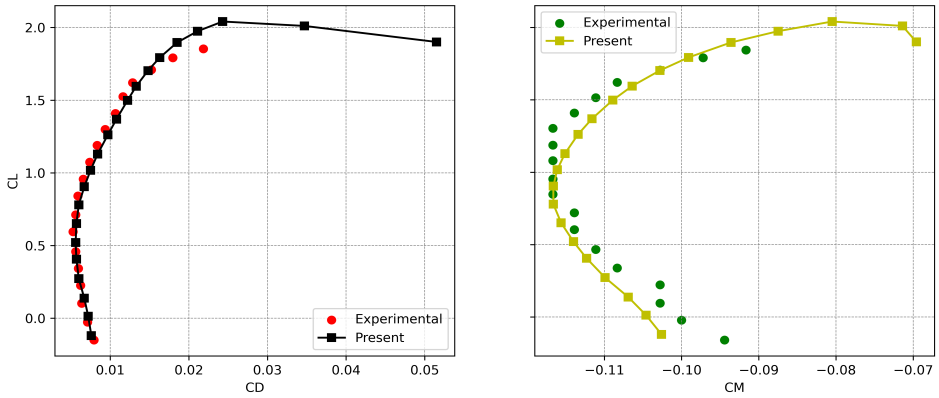

Figure 5.4: *NLF(1)-0416 Airfoil Case. C_D (left) and C_M (right) across a range of angles of attack. On the left, the data also fit very well with each other before the stall effect. On the right, where the C_M about the quarter chord point is plotted, a relatively small difference can be seen.*

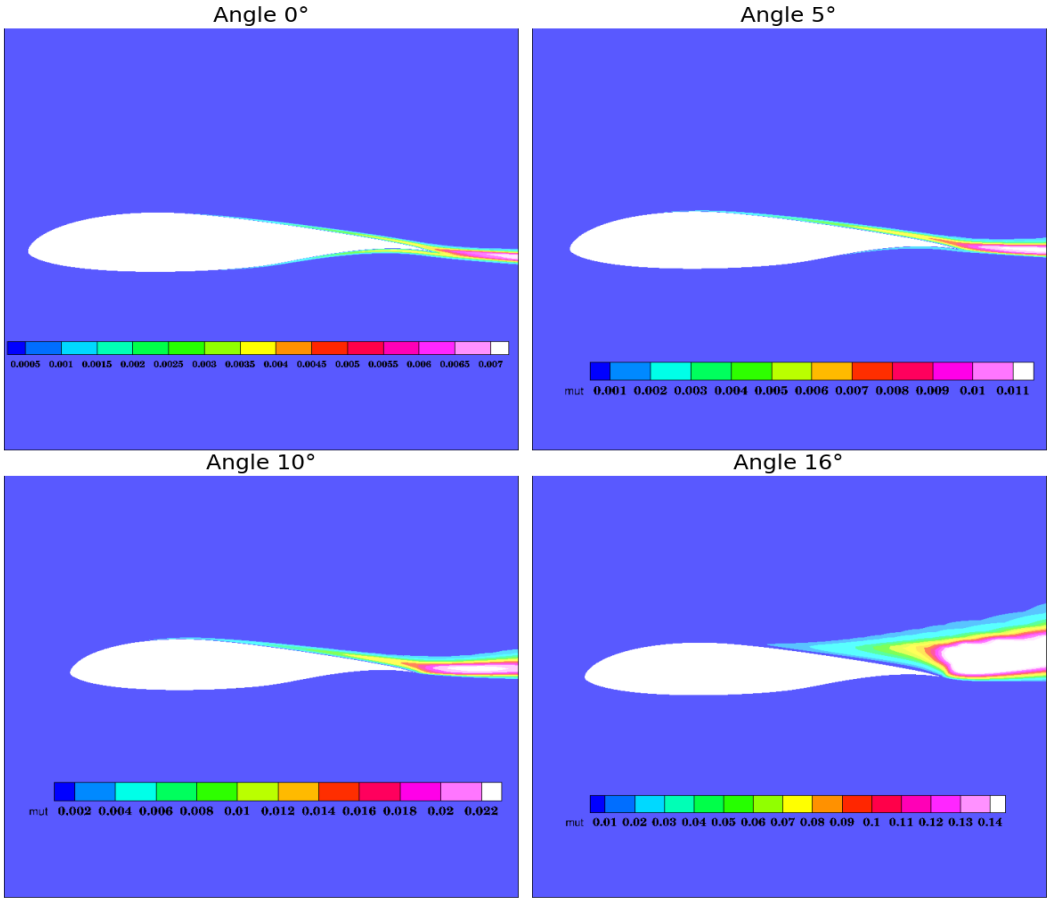

Figure 5.5: *NLF(1)-0416 Airfoil Case. Turbulent viscosity (μ_t) fields at four angles of attack for $M_\infty = 0.3$ and $Re_c = 6 \cdot 10^6$. The μ_t fields show where the transition from laminar to turbulent flow occurs. By increasing the AoA, especially from 5° to 16°, the μ_t increases at the rear half of the Airfoil and the turbulent zone becomes wider.*

5.1.2 Case Description

The following studies are dealing with the NLF(1)-0416 airfoil for two different flow conditions, as in Table 7.3. F1 stands for "Flow 1" and F2 for "Flow 2". The two subcases are presented and discussed simultaneously.

Table 5.1: *NLF(1)-0416 Airfoil Case. Flow conditions*

Flows	$\mathbf{M_\infty}$	$\mathbf{a_\infty}$	$\mathbf{Re_c}$
F1	0.1	2.03^o	4×10^6
F2	0.3	4.07^o	6×10^6

The turbulence intensity is Tu=0.15% and the maximum dimensionless wall distance $y^+ = 0.74$, in both cases.

Initially, aiming to create the training DB, for each flow condition, the Latin Hypercube Sampling (LHS) method was used to sample 40 vectors (with 4 entries each) from the uncertain variable's space. Next, the RANS solver computed the flow and two training DBs were created. The normalized input features are visualized in a parallel coordinates plot (figure 5.6). The latter provides a qualitative understanding of the relationships between all datapoints.

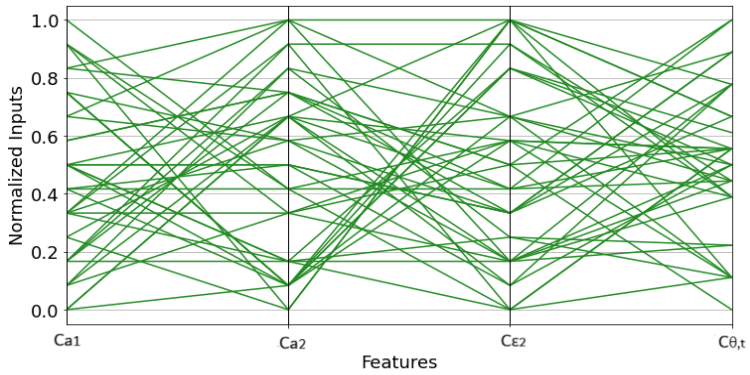

Figure 5.6: *NLF(1)-0416 Airfoil Case. The effectiveness of Latin Hypercube Sampling method in covering the input space can be observed.*

From the 40 Training Patterns in both flow conditions, C_L and C_D are plotted together and presented in figure 5.7.

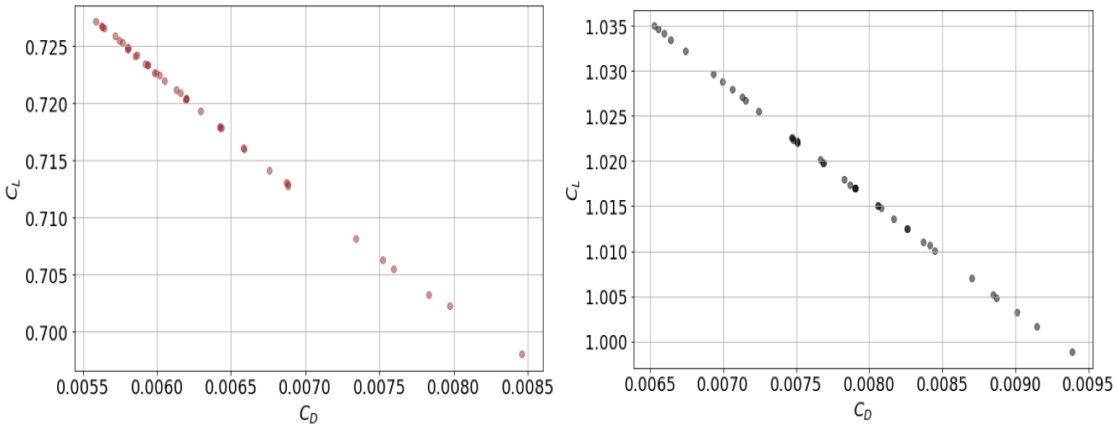

Figure 5.7: *NLF(1)-0416 Airfoil Case. C_L vs C_D plots for F1(left) and F2(right). One may observe that the two quantities vary inversely.*

Figure 7.7 illustrates the skin friction coefficient (C_f) distributions, plotted for the nominal values of the uncertain variables along with the ones produced by the 40 additional samples. The steep lines in this plot correspond to transition points from laminar to turbulent flow. One may easily see that small changes in the uncertain variables shift the transition onset considerably. The latter highlights the great sensitivity of the computed results depending on the aforementioned empirical constants to be used in the $\gamma - \tilde{Re}_\theta$ transition model.

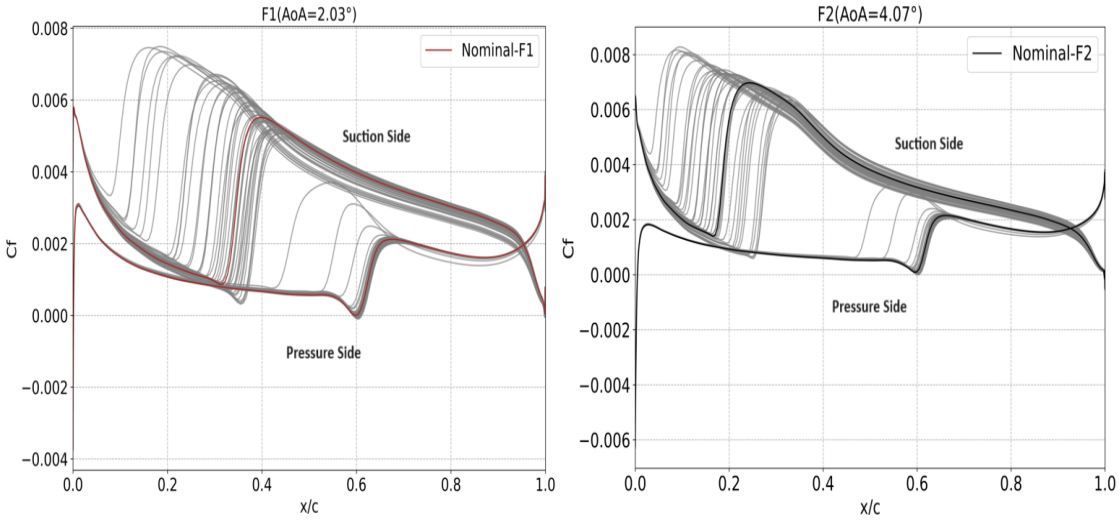

Figure 5.8: *NLF(1)-0416 Airfoil Case. C_f distributions for both flow conditions. Rather small changes in the uncertain variables pose a great impact on the transition point. In the second case a greater turbulent region around the Airfoil section can be observed. The latter primarily occurs due to the higher angle of attack.*

5.2 The Low Cost Surrogates

DNNs were trained separately for the two flow conditions, and their performance was checked on a sizeable unseen dataset. The aim was, for each case, to obtain DNNs that could predict the C_L and C_D of the airfoil, receiving as inputs the aforementioned four uncertain variables.

5.2.1 Datasets and Preprocessing

The following lines further analyze the data used for the training, retrieve additional insights, and present how the testing phase was carried out.

Training

Along with the first DB of 40 TP, on which every model of this chapter was finetuned (selection of architecture and hyperparameters), four more DBs of different sizes were also generated using LHS. This step was taken to examine the generalization capabilities of model configurations optimized for a specific DB, when trained on different (size-wise) ones. This process was carried out for both flow conditions. The additional DBs were all independent of each other (meaning that the smaller ones were not subsets of the larger ones) and were composed by 20, 30, 50, and 60 TP, respectively. It is underlined that all models were finetuned in the DB of 40TP, solely for F1, and that all the additional DBs were used to train the predetermined DNN configurations, and check their performance. Regarding the RBFN surrogates, from each different DB, one two-output RBFN was trained. Concerning the DNN models, the following dilemma arises for each DB: making one DNN predict the two QoIs or two separate DNNs for the same task. In this chapter, two separate DNNs were created for the same task.

First, to better understand the data, a check for correlations among them was carried out. The DBs of 40, 50, and 60TP were selected arbitrarily out of all the DBs, and only for them, correlation matrices were generated, illustrating the Pearson Correlation Coefficient (PCC) (explained in appendix A.1.3)) between all features in each DB. Even though PCC assumes only linear relationships, it is presented as a tool that could indicate some less relevant features or possible noise in the data. These features could be dropped to make the model(s) less expensive or even more accurate. For simplicity, the matrices are visualized as colored heatmaps (figure 5.9), where colors are in complete accordance with the PCC values.

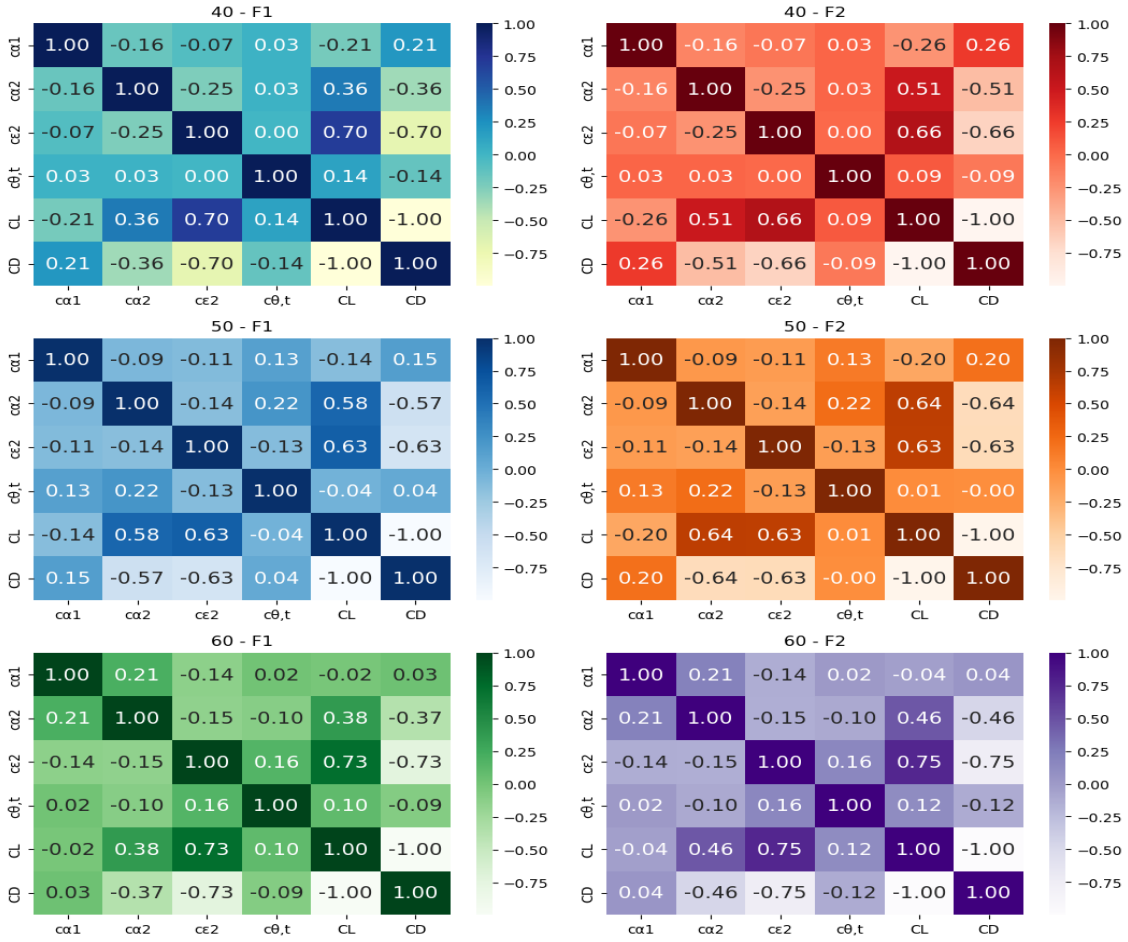

Figure 5.9: *NLF(1)-0416 Airfoil Case. PCCs in heatmaps for the DBs of 40, 50, and 60TP. F1 on the right and F2 on the left. A consistent pattern was observed. The most substantial linear influence in the QoIs was provided by the $c_{\varepsilon 2}$ and c_{a2} constants, and this is true because of the higher absolute PCC values one may easily see by looking at the output columns across all heatmaps (except the value of 1.00 that displays the PCC between the outputs themselves). C_{a1} and $C_{\theta,t}$ displayed minimal PCC values regarding the QoIs and, thus, no linear correlations. This observation might be related to "Generally, $c_{\varepsilon 2}$ and c_{a2} are the key parameters of the $\gamma - \tilde{Re}_\theta$ model." stated in [17].*

Lastly, it must be noted that after the DBs were non-dimensionalized, the fitted scaling models (known as scalers in Python) employed for the latter were saved for later use, coupled with DNNs trained on the DB where the scaler was fitted. Each scaler is used every time the trained DNN is called to scale every data point before it gets fed into the model and, also, to unscale back the model's predictions, using information derived from the initially available DBs, used for training.

Testing

As done in [54], another 300 value-sets of the 4 uncertain variables, were randomly generated (following the normal distribution) and evaluated using both the surrogates and the high-fidelity RANS solver (figures 5.10 and 5.11), aiming to assess the performance of each trained model on this larger unseen dataset.

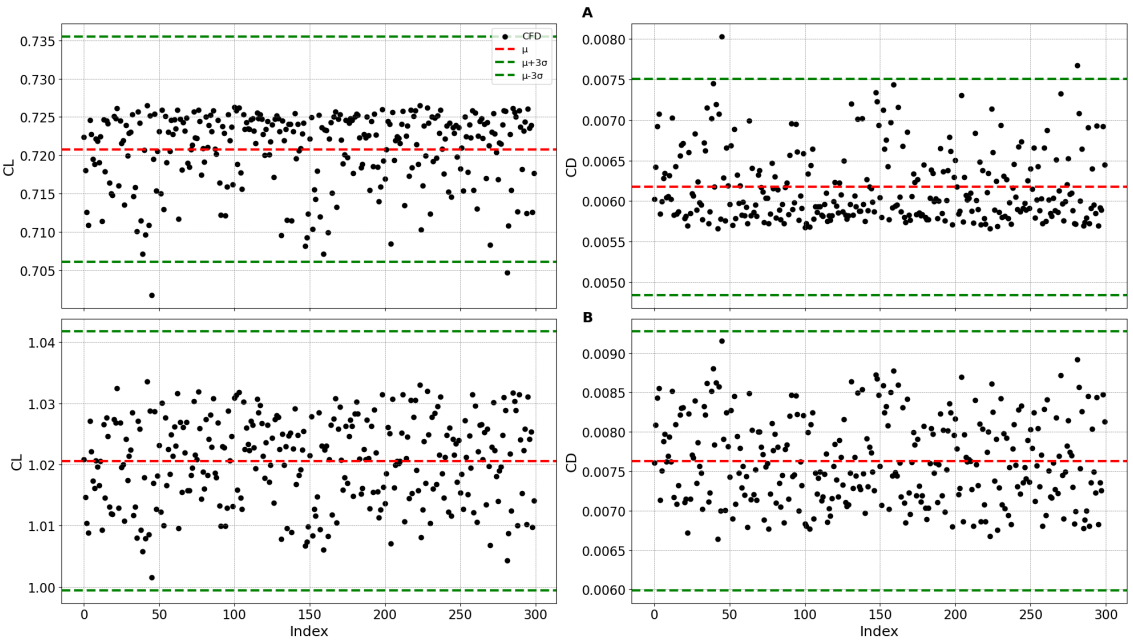

Figure 5.10: *NLF(1)-0416 Airfoil Case. Sampled out of the normal distribution, the 300 uncertain vectors, computed by the CFD, provided the C_L and C_D results for both flow conditions. In F1, both QoIs appear to display a downward and upward trend, respectively. Additionally, only 2 out of the 300 data points were observed to fall outside the interval of [μ-3σ,μ+3σ] in F1 and, despite the more significant variance of the data, in F2, all data points fell inside the interval under consideration.*

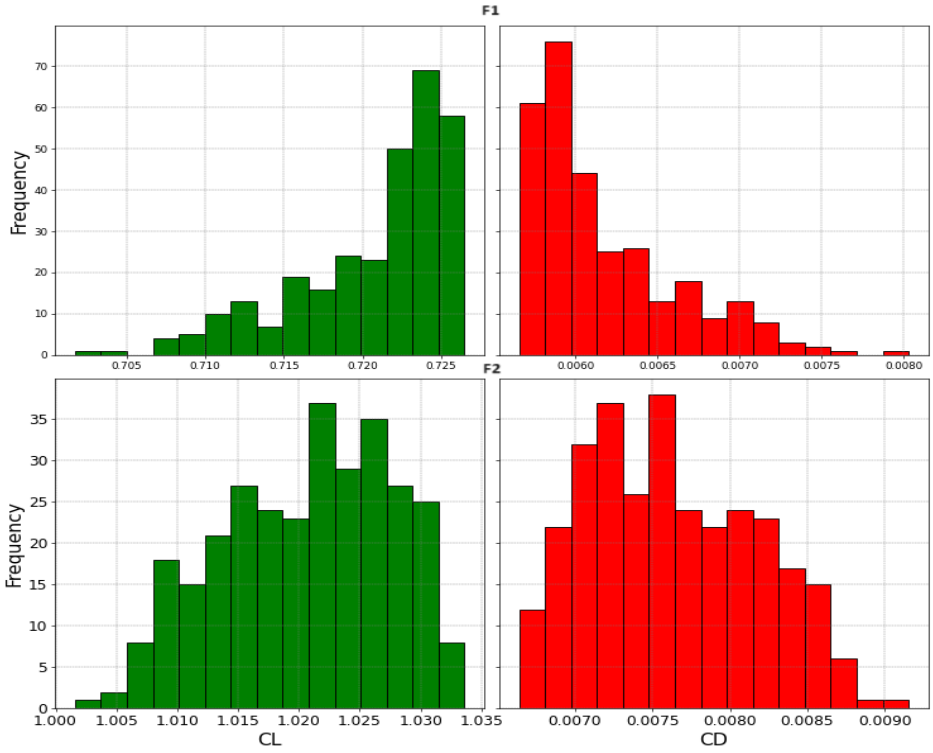

Figure 5.11: *NLF(1)-0416 Airfoil Case. The 300 CFD-computed C_L and C_D distributions for both subcases. The broader distribution of F2 results is evident here as well. The QoIs of F2 display a distribution resembling more to the normal one compared to F1 results, which seem to be right and left-skewed distributions, respectively.*

5.2.2 DNN Configuration and Metrics

Initially, through trial and error, many DNN configurations were manually created. The objective behind this approach was to rapidly test such a model's capabilities, aiming to, later on, deliver the task of finding even better configurations to the optimization software EASY. All models were trained in TensorFlow using the model checkpoint callback to save the model that produced the lowest validation loss across all epochs. Every DNN configuration was fine-tuned regarding the F1 and then was, also trained in F2. The training phase was carried out for 1500 epochs, and, as regards the computational cost, every model was trained on a GPU RTX3060 for less than a minute. Thus, the computational cost for training the two models is less than two minutes. Considering the cost of creating the DB that is equal to 40TP, using the RANS solver, is about 10 hours, in this particular scenario, the DNNs' training costs can be viewed as trivial. Moreover, the validation split was selected at 0.1, meaning that 90% of each DB was used for training and 10% for validation. Lastly, the batch size was set to match the entire dataset, as changes in this parameter were observed to have minimal impact on the model's accuracy;

rather, this was due to the small DB sizes. The first best-performing configuration was the following:

Table 5.2: *NLF(1)-0416 Airfoil Case. Manually-found DNN configuration*

Layers	Neurons	Activation Function	Batch Size	Loss
7	(4, 128, 64, 32, 64, 1)	ReLU/tanh	Training Patterns	MAE

This architecture proved to work well in predicting both C_L and C_D. Overall, four models were trained separately, with two models for each flow condition (F1 and F2). Their corresponding learning curves are presented in figure 5.12 and their error metrics in tables 5.3 and 5.4.

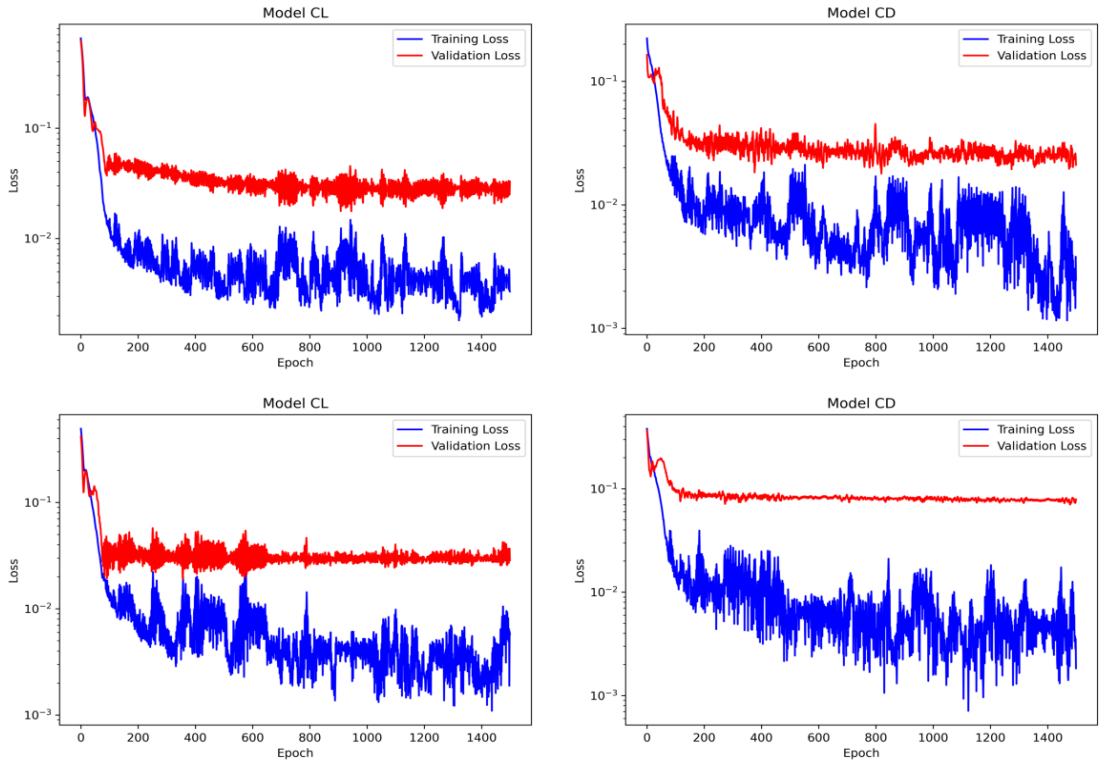

Figure 5.12: *NLF(1)-0416 Airfoil Case. Learning curves of the first well-performing manually created DNNs. The upper plots concern F1 and the lower ones F2. Models for: C_L on the left and C_D on the right.*

Table 5.3: *NLF(1)-0416 Airfoil Case. DNN training phase error metrics*

DNN	Train Loss (MAE)	Validation Loss (MAE)
F1-C_L	0.0132	0.0176
F1-C_D	0.00613	0.01786
F2-CL	0.0199	0.0189
F2-C_D	0.00645	0.07023

Table 5.4: *NLF(1)-0416 Airfoil Case. DNN testing error metric*

DNN	Test Loss (MAPE)
F1-C_L	0.059%
F1-C_D	0.38%
F2-C_L	0.085%
F2-C_D	0.89%

It is evident (at least for most of the models in figure 5.12) that, around 200 epochs, the validation loss stopped decreasing. This means that if the training had stopped at about that point, the results would still be of the same order of magnitude. Additionally, as regards the testing dataset, it was decided to display the MAPE error metric, aiming for a more interpretable model measure. The C_L coefficient indicated a more accurate approximation by roughly ten times, compared to the C_D across both flow conditions. This observation was also consistent with the RBFNs in [54]. However, for both QoIs, the testing error metric seems very satisfactory.

Next, the optimization software was exploited by being provided with the previous well-performing architecture as an initial candidate solution. The aim was to find, across the same amount of epochs, DNN architectures that would produce minimum validation losses. Hence, the validation loss metric (monitored by model checkpoint) was set as the objective function of this **hyperparameter optimization**.

The **design variables** were: the **activation function types**, one for every hidden layer and one for the output layer, the **number of hidden layers**, and the **number of neurons per layer** as a power of 2.

The DNN architecture presented in table 5.5 was found by setting EASY to search for the best model C_D. However, this resulting architecture also proved to perform well in predicting both QoIs, so it is illustrated. As regards the search for the best model C_L, EASY produced some, only slightly better than the latter, architectures, which, though, displayed significantly lower performance in predicting C_D. Thus,

they were not considered worthy of inclusion in this thesis. It's important to point out that manual and EASY-found DNN architecture results were of the same order of magnitude and not very far from each other. Nevertheless, the architecture found by EASY was better than the manually created one in every error metric (44-82% better). The final statement is confirmed by observing their test losses (tables 5.4 and 5.7) along with their learning curves, where in the ones in figure 5.13 (representing the EASY-found DNNs), the validation losses delay noticeably in reaching the plateau. Last, regarding the optimization cost, it must be noted that to optimize a single configuration, EASY needs around 6 hours, which was paid only once for F1, as stated above.

Table 5.5: *NLF(1)-0416 Airfoil Case. Easy-found DNN configuration*

Layers	Neurons	Activation Function	Batch Size	Loss
8	(4, 256, 512, 4096, 64, 32, 512, 1)	GELU/RelU	Training Patterns	MAE

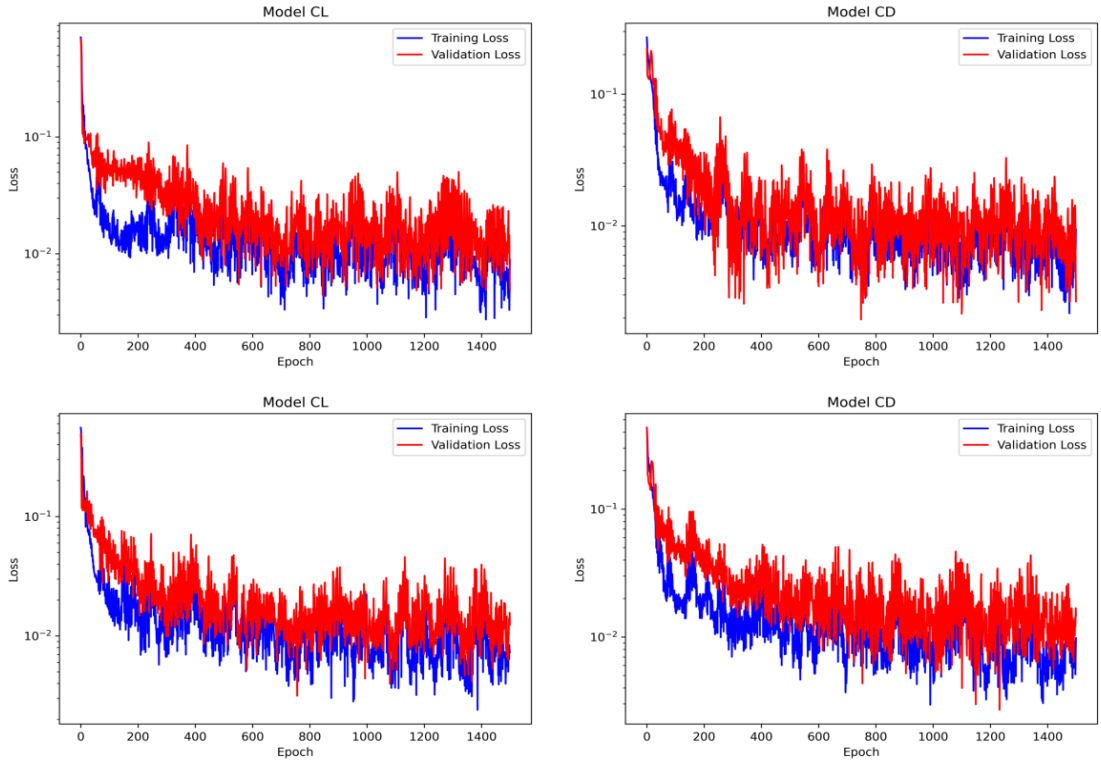

Figure 5.13: *NLF(1)-0416 Airfoil Case. Learning curves of the architecture that EASY found. The upper plots are for F1 and the lower ones for F2. Models for: C_L on the left and C_D on the right.*

Table 5.6: *NLF(1)-0416 Airfoil Case. DNN training phase error metrics*

DNN	Train Loss (MAE)	Validation Loss (MAE)
F1-C_L	0.00517	0.00438
F1-C_D	0.00647	0.00195
F2-C_L	0.00519	0.00315
F2-C_D	0.01433	0.00269

In table 5.6, it can be seen that some of the models displayed a lower validation loss than the train loss. The latter can be considered a totally random event, probably due to the limited validation data. As regards the testing metrics of table 5.7, a complete consistency with the ones in the previous manual architecture (table 5.4) can be noticed.

Table 5.7: *NLF(1)-0416 Airfoil Case. DNN testing error metric*

DNN	Test Loss (MAPE)
F1-C_L	0.021%
F1-C_D	0.2%
F2-C_L	0.015%
F2-C_D	0.5%

Additionally, aiming to compare the above (EASY-found) best-performing DNNs with the existing RBFNs trained with an identical DB of 40 TP, it was decided to present their corresponding prediction vs actual value plots, one on top of the other. The DNNs seem superior across all QoIs. (figure 5.14).

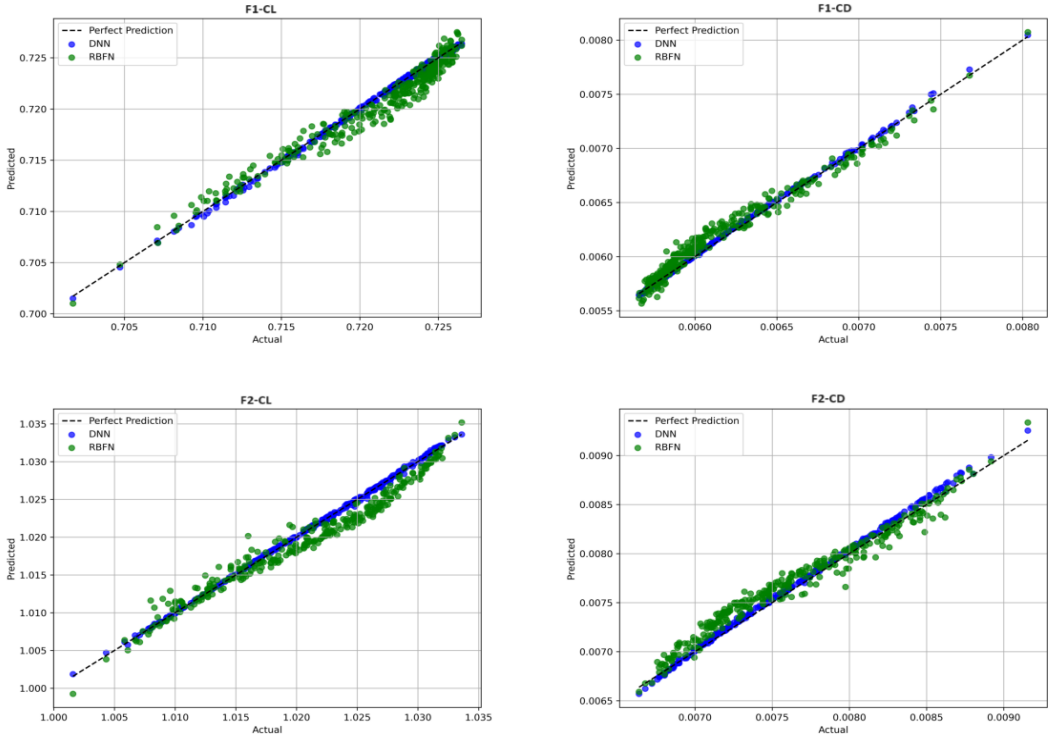

Figure 5.14: *NLF(1)-0416 Airfoil Case. Prediction vs. actual value plots for DNNs and RBFNs. The upper plots concern F1 while the lower ones F2 (namely the two flow conditions). The plots showcase the divergence of each model's predictions from the perfect predictions. The two one-output DNNs are placed on top of the corresponding RBFN for both cases. The DNNs outperform the RBFNs consistently across every QoI.*

5.2.3 DNN and RBFN Test Metrics Across Different DBs

Next, DNN and RBFN configurations are utilized to train models across all the aforementioned DBs. The models are also checked in DB-300. To clearly understand each model's performance and compare it to each other, it was decided to present their testing error metric (MAPE) side by side.

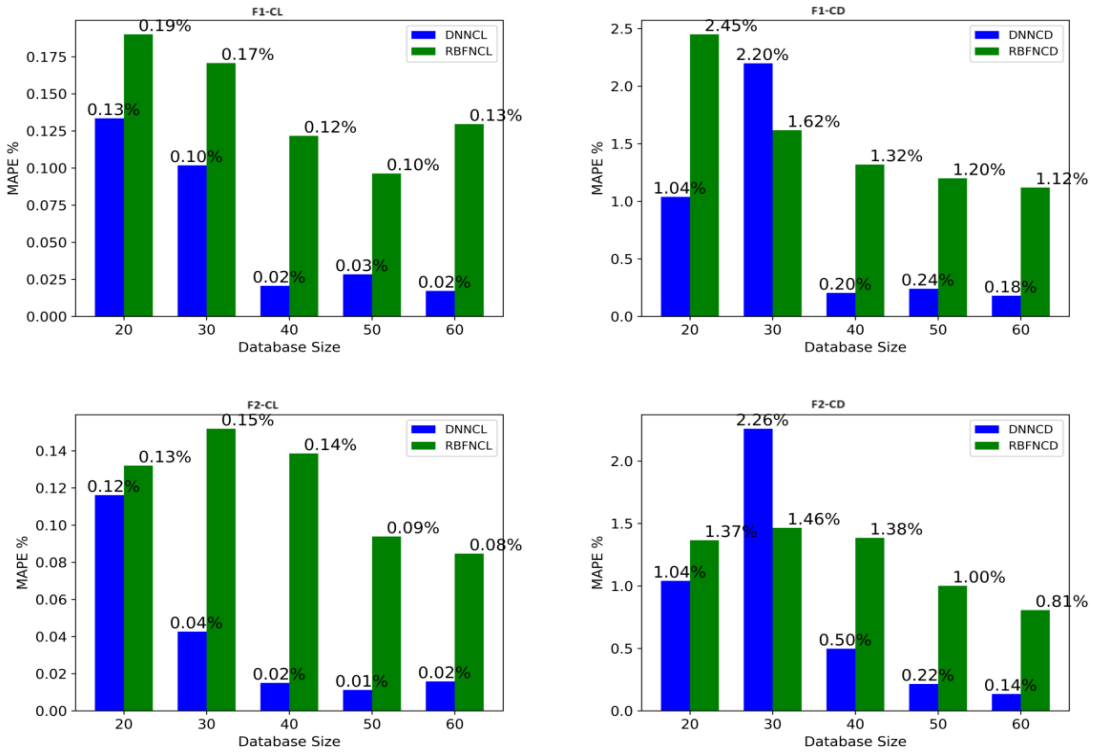

Figure 5.15: *NLF(1)-0416 Airfoil Case. DNN vs. RBFN testing error metric. All models of each subcase were trained in the five aforementioned DBs and tested on the same DB of 300 samples. Upper plots concern F1 and lower ones F2. The superiority of the DNN predictions is obvious.*

In figure 5.15, the models generally display a decreasing tendency in their test error metric as the number of TPs increases, as expected. This observation is not universal, though, as regards the smaller DBs. More precisely, the DB of 30TP is noticed to produce by far the worst DNN model C_D. These two DNNs are the only ones to get outperformed by the RBFN's C_D predictions. The rest of the DNNs, especially the ones trained on the larger DBs, demonstrate excellent performance, placed next to the RBFNs. For the DBs of 40, 50, and 60TPs, the DNN's results are about one order of magnitude better than those produced by the RBFNs.

5.3 UQ with DNN

By employing the DNNs trained on the DB of 40TP as surrogates, all UQ methods described in chapter 4 are carried out. The total cost of the surrogates is 40 TU, which is paid upfront (DB creation). The usage of these models is essentially free, which allows the UQ-DNN methods to be conducted for several samples without concern for the cost. Due to its large demands in computations, the Monte Carlo

method is only carried out with the surrogate and for the previously CFD-computed testing DBs of 300 samples (a generally insufficient number of samples for such a method). The UQ-DNN and UQ-CFD comparison is mainly focused on the non-intrusive, Gauss Quadrature, and Regression-Based PCE methods.

5.3.1 Monte Carlo with DNN

Starting from 10^3 random samples w.r.t normal distribution, the MC method's sensitivity to sample size is checked. In tables 5.8 and 5.9, it is presented that for both subcases, increasing the sample size provides no accuracy improvement since the results for sample sizes up to 10^5 are almost identical. Thus, the method is additionally conducted for significantly smaller sample sizes to manifest the sample size that would be considered insufficient.

Table 5.8: *NLF(1)-0416 Airfoil Case. MC method for F1:*

Method/Tool	**Time Units**	μC_L	σC_L	μC_D	σC_D
MC-DNN(10^5)	40	0.7210	0.004965	0.006164	0.0004531
MC-DNN(10^4)	40	0.7210	0.004924	0.006161	0.000449
MC-DNN(10^3)	40	0.7210	0.005054	0.006164	0.0004628
MC-DNN(500)	40	0.7208	0.004933	0.006179	0.0004484
MC-DNN(250)	40	0.7205	0.004991	0.006203	0.0004537
MC-DNN(125)	40	0.7210	0.005277	0.006158	0.0004826
MC-CFD(300)	300	0.7208	0.004904	0.006185	0.0004792

Table 5.9: *NLF(1)-0416 Airfoil Case. MC method for F2:*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
MC-DNN(10^5)	40	1.0210	0.007038	0.007638	0.0005671
MC-DNN(10^4)	40	1.0211	0.007035	0.007634	0.0005668
MC-DNN(10^3)	40	1.0211	0.007163	0.007632	0.0005768
MC-DNN(500)	40	1.0206	0.006888	0.007669	0.0005546
MC-DNN(250)	40	1.0203	0.006968	0.007696	0.000563
MC-DNN(125)	40	1.0213	0.007276	0.007614	0.0005903
MC-CFD(300)	300	1.0206	0.00705	0.007632	0.0005473

Only tiny fluctuations are evident in both flows across all QoIs, even for the 250 samples. Thus, it's concluded that the present problem doesn't require considerable datasets to quantify its uncertainties using the MC method and that the MC-DNN provides very close results as the MC-CFD. As regards the MC conducted for the 125 samples, the standard deviation values start to diverge noticeably.

5.3.2 UQ using PCE

Gauss Quadrature PCE

First, gPCE-CFD for k = 2 is conducted, and its results are used as a reference. Next, gPCE-DNN is carried out for chaos orders of k = 2, 3, and 4. This PCE-cost expression $(k+1)^M$, which provides the number of objective function evaluations needed according to chaos order and the number of uncertain variables, is the reason why gPCE-CFD is avoided for higher chaos orders and these were performed only with the surrogate.

Table 5.10: *NLF(1)-0416 Airfoil Case. gPCE method for F1*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
gPCE-CFD($k=2,\ 81$)	81	0.7210	0.004923	0.006153	0.0004477
gPCE-DNN($k=2,\ 81$)	40	0.7210	0.005002	0.006161	0.0004573
gPCE-DNN($k=3,\ 256$)	40	0.7210	0.004939	0.006163	0.0004511
gPCE-DNN($k=4,\ 625$)	40	0.7210	0.004973	0.006163	0.0004534

Table 5.11: *NLF(1)-0416 Airfoil Case. gPCE method for F2*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
gPCE-CFD($k = 2,\ 81$)	81	1.0210	0.006955	0.007603	0.0005399
gPCE-DNN($k = 2,\ 81$)	40	1.0211	0.006978	0.007634	0.0005648
gPCE-DNN($k = 3,\ 256$)	40	1.0210	0.007055	0.007636	0.0005682
gPCE-DNN($k = 4,\ 625$)	40	1.0210	0.007052	0.007636	0.0005683

In tables 5.10 and 5.11, it is evident that gPCE-DNN approximate the gPCE-CFD results across all QoIs excellently. The increase of chaos order for the gPCE-DNN present roughly the same results as k=2. The σ_{C_D} approximations deviate slightly. These deviations are negligible but highlighted due to the superior quality level in the rest of the predictions.

Regression-Based PCE

Next, rPCE is carried out with a chaos order of k = 2. For this method, the 81 CFD-evaluated GQ nodes are assessed for generating the method's high-fidelity results. Using the LHS method, three additional datasets of 70, 140, and 280 samples are generated and evaluated with the DNNs. The rPCE-DNN is conducted for these datasets. It is reminded that, for the rPCE with k = 2 and M = 4, the minimum number of samples needed for the method to be feasible is equal to the number of PCE coefficients that need to be evaluated. Namely, $\frac{(3+4)!}{3!\cdot 4!} = 35$.

Table 5.12: *NLF(1)-0416 Airfoil Case. rPCE method for F1*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
rPCE-CFD($k = 2,\ 81$)	81	0.7208	0.004584	0.006174	0.0004199
rPCE-DNN($k = 2,\ 70$)	40	0.7207	0.004574	0.006231	0.0004271
rPCE-DNN($k = 2,\ 140$)	40	0.7204	0.0044	0.006236	0.0004196
rPCE-DNN($k = 2,\ 280$)	40	0.7205	0.004415	0.006194	0.0004227

Table 5.13: *NLF(1)-0416 Airfoil Case. rPCE for F2*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
rPCE-CFD($k = 2,\ 81$)	81	1.0209	0.00634	0.007608	0.0004929
rPCE-DNN($k = 2,\ 70$)	40	1.0198	0.005464	0.007685	0.0004631
rPCE-DNN($k = 2,\ 140$)	40	1.0207	0.005712	0.007649	0.0004329
rPCE-DNN($k = 2,\ 280$)	40	1.0207	0.005519	0.007653	0.0004521

The results of rPCE-DNN (tables 5.12 and 5.13) are acceptable but not quite as good as the ones of the gPCE-DNN. The standard deviation values of the rPCE-DNN, in F2, are deviating noticeably from the rPCE-CFD ones. Lastly, the rPCE-DNN proves insensitive to the increase in the oversampling ratio, and mild changes over the different sampling sizes occur primarily due to the stochasticity of the LHS.

5.4 Aggregated Results

To summarize the present study, the most valuable insights from the conducted UQ methods are chosen to be presented side by side in tables 5.14 and 5.15. By doing so, one may observe a more comprehensive comparison between the different UQ methods.

Table 5.14: *NLF(1)-0416 Airfoil Case. UQ methods for F1*

Method/Tool	Time Units	μC_L	σC_L	μC_D	σC_D
MC-DNN(500)	40	0.7208	0.004933	0.006179	0.0004484
gPCE-DNN($k = 2,\ 81$)	40	0.7210	0.005002	0.006161	0.0004573
rPCE-DNN($k = 2,\ 140$)	40	0.7204	0.004310	0.006236	0.0004196
MC-CFD(300)	300	0.7208	0.004904	0.006185	0.0004792
gPCE-CFD($k = 2,\ 81$)	81	0.7210	0.004923	0.006153	0.0004477
rPCE-CFD($k = 2,\ 81$)	81	0.7208	0.004584	0.006174	0.0004199

Table 5.15: *NLF(1)-0416 Airfoil Case. UQ methods for F2*

Method/Tool	**Time Units**	μC_L	σC_L	μC_D	σC_D
MC-DNN(500)	40	1.0206	0.006888	0.007669	0.0005546
gPCE-DNN($k = 2,\ 81$)	40	1.0211	0.006978	0.007634	0.0005648
rPCE-DNN($k = 2,\ 140$)	40	1.0207	0.005712	0.007649	0.0004329
MC-CFD(300)	300	1.0206	0.00705	0.007632	0.0005473
gPCE-CFD($k = 2,\ 81$)	40	1.0210	0.006955	0.007603	0.0005399
rPCE-CFD($k = 2,\ 81$)	81	1.0209	0.00634	0.007608	0.0004929

As mentioned above, all UQ methods carried out with the surrogates produce very satisfactory results. However, the key conclusions regarding the NLF(1)-0416 case are drawn and presented in the following points.

- One DNN-training costs less than 1/15 TU, considering that a single CFD run needs around 15 minutes on the GPU RTX3060, while the training of a single DNN requires less than 1 minute on the same GPU.
- Taking into account the 300 CFD evaluated samples provide an accurate MC prediction, this method is the closest to what could be called "ground truth". Nevertheless, these 300 CFD evaluations may not be feasible in a real-life, more computationally expensive problem.
- The gPCE method outperformed the rPCE with CFD and DNN as evaluation tools.
- After the MC, the next best choice is the gPCE method (for k=2), which, coupled with the CFD solver, costs 81 TU. By employing the surrogate DNNs for UQ, instead of performing the gPCE-CFD(81), the computational cost is cut down to more than 50%, recalling that the DNN prediction errors can be considered negligible.
- Considering a RDO scenario using evolutionary algorithms (which usually demand large numbers of objective function evaluations), this 50% cost reduction in the task of UQ, namely in every iteration, can drastically reduce computational cost.
- Lastly, it is essential to highlight that the present two separate surrogate models (C_L and C_D DNNs), compared with the previously described RBFN for the same task, have shown to reduce the prediction error for both QoIs by approximately ten times; the price to pay for this is complexity and higher training time (considering the training time of RBFNs is less than 5 seconds).

5.5 Additional Studies on the Surrogate Model

As discussed earlier, the present DNNs performed sufficiently for predicting C_L and C_D. Their accuracy proved much better than that of RBFNs, yet with the cost of an even increase in training time. Two additional studies regarding the surrogate model are conducted. The first is to combine the DNN with RBFN through the stacking ensemble technique, and the second is to train the DNN by dropping (leaving out) the possibly less important input features (according to the PCC indications).

5.5.1 Stacking Ensemble

As described in chapter 2, stacking is an ensemble technique that allows the combination of different types of models, leveraging each model's strengths and weaknesses in a way that aims to create a new, more accurate, from every base learner, model. The pre-trained DNNs and RBFNs are stacked using a linear regression ML model as the meta-learner. In order to do so, the new model (Linear Regression) receives the predictions of the base models regarding the training DBs as inputs and, identically to the base models, the QoIs as target outputs. The simple linear regression ML model (meta-learner) is trained, and its performance is visualized in figure 5.16. It is highlighted that, as in figure 5.15, the models are trained on the DBs of 20, 30, 40, 50, and 60TP for the two different flow conditions (F1 and F2). Their performance is presented through their MAPE error metric (figure 5.16) computed by their predictions on the large unseen DB of 300 random samples.

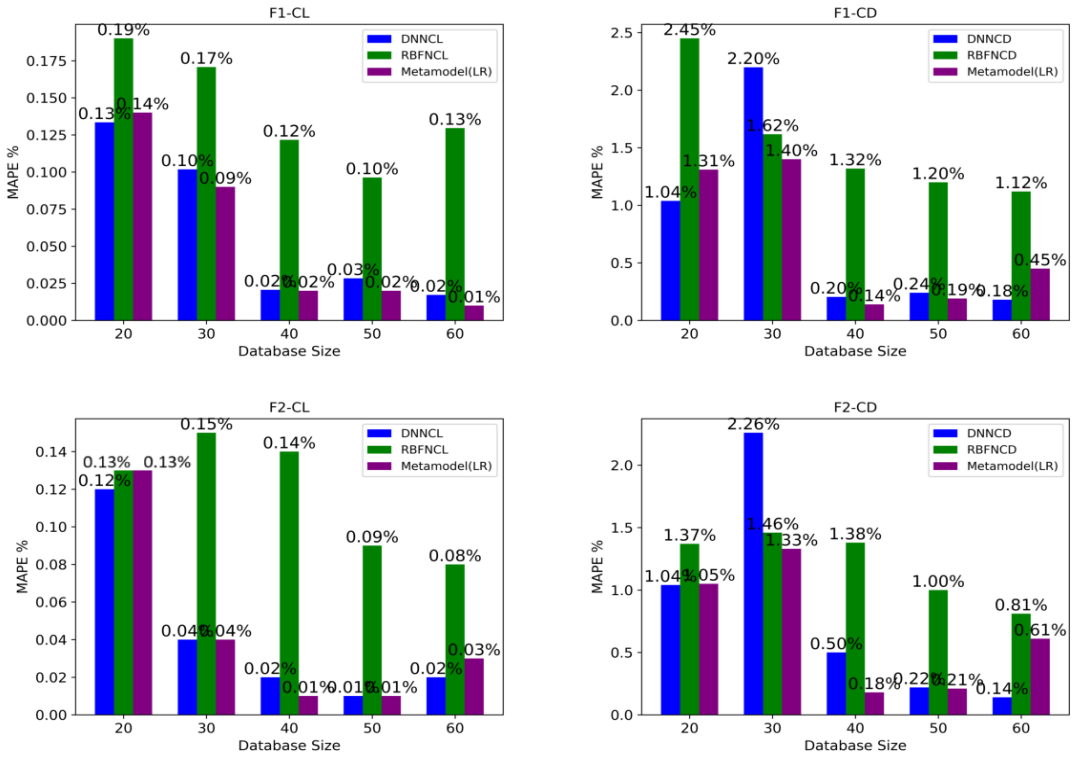

Figure 5.16: *NLF(1)-0416 Airfoil Case. DNN vs. RBFN vs. Metamodel(LR) testing error metric. All models were trained on the five DBs above and tested on the same DB of 300 samples. Upper plots concern F1 and lower ones F2. In most cases (around 3 out of 5), the ensemble provided improved accuracy. However, in some other cases, it performed worse than the DNN.*

The ensemble's underperformance is observed only in the DBs of 20 and 60TP. Regarding the "middle" DBs (30, 40, and 50TP), the ensemble's performance is equal to, or better than the most accurate base model (DNN). Regarding the computational cost, it is evident that since stacking requires training multiple models, the training cost must be increased. However, in the present case, the cost of all surrogates is very low compared to the cost of a CFD run (1TU). By comparing the surrogates solely with each other (all trained on the RTX3060 GPU), as mentioned above, each DNN training cost is around 20-60 seconds, while the costs of the RBFN and LR are less than 5 seconds each. These numbers can lead to the assumption that the costs of RBFN and LR are completely trivial and that the cost of the DNN determines the total cost of training the ensemble.

5.5.2 Feature Selection

According to the indications shown by the PCC values in figure 5.9, the best performing DNN configuration (table 5.5) was trained, having removed from the input

features the possibly "irrelevant ones". The study was carried out for F1 and F2 using solely the DB of 40TP for training, and the DB of 300 samples for testing. Three scenarios were checked. Either leaving out from the input feature space: C_{a1} or $C_{\theta,t}$ and by leaving out both C_{a1} and $C_{\theta,t}$.

For reference purposes these three scenarios were named as follows:

- Test1: DNN training leaving out c_{a1}
- Test2: DNN training leaving out $c_{\theta,t}$
- Test3: DNN training leaving out c_{a1} and $c_{\theta,t}$

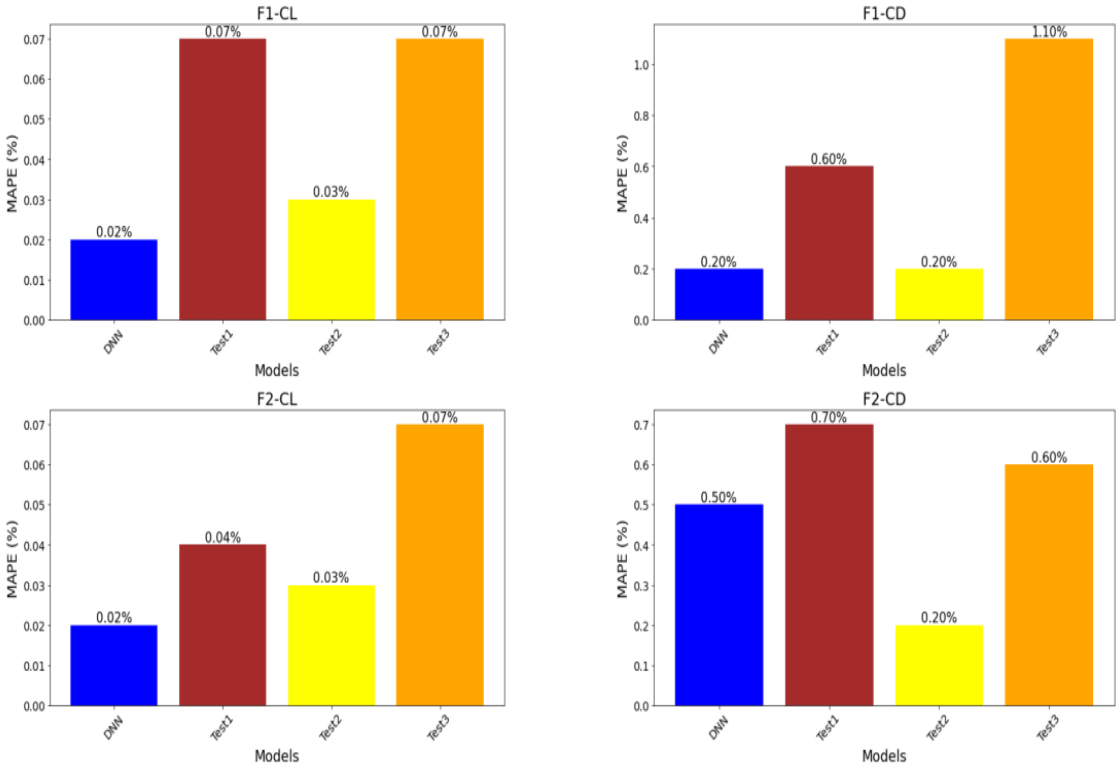

Figure 5.17: *NLF(1)-0416 Airfoil Case. The original DNN test metric is compared with the ones of Test1, Test2 and Test3 for F1 (upper plots) and F2 (lower plots).*

One may easily see (figure 5.17) that the DNNs trained with fewer features generally performed worse than the original DNN. However, the performance of these smaller models seems very close to the original one. Test2 provided, by far, the best results across the three tests. Moreover, in the C_D models, Test2 provided equal and superior accuracy metrics from the original DNN regarding F1 and F2, respectively. The training times of the smaller models were obviously lower (Test3 had around 20% faster training time than the original DNN). Despite the trivial training times of the surrogates, involved in this thesis, the latter could become very useful in larger applications where models can be extremely large and training times very costly.

5.5.3 Conclusions

- It seems that the best surrogate (DNN) prediction accuracy is further improved, even up to 50%, by combining it with the RBFN through the stacking ensemble technique. The only limit to how many models can be stacked together is the computational cost of training the base models and the meta-model, which are determined by the problem at hand.
- It is demonstrated that using PCC indications to reduce the features of a DNN (feature selection) achieved smaller and up to 20% faster trainable models whose performance is close to the original model, and also, one model that appeared even better (60%) than the original in predicting the C_D. In cases where training DNNs becomes costly, this approach can be very handful.

Chapter 6

UQ in a Wing Case

This chapter deals with using DNNs for UQ in the case of a 3D flow around an isolated wing. Over and above the transition model's four constants, the surface roughness, $h_{rms} \sim \mathcal{N}(5 \cdot 10^{-6}, 1.6 \cdot 10^{-6})$, was included as the fifth uncertain variable (M=5). This quantity can be characterized of questionable fidelity as well, mainly due to measurement limitations. The aim was to quantify the impact of uncertainties on the C_L and the C_D, using DNNs.

6.1 The ONERA M6 Wing

6.1.1 Case Description

The primal problem concerns a flow around the ONERA M6 wing at $M_\infty = 0.262$, $Re = 3.5 \cdot 10^6$, angle of attack and yaw angle both set to 0^o. Spalart-Allmaras and the $\gamma - \tilde{Re}_\theta$ model were used for turbulence and transition modeling, respectively. The inlet turbulence intensity is Tu=0.2%. The available evaluated 120 uncertain vectors are obtained from the LHS. The data is organized in two DBs: one with 80 TP and the second with all the available TP, namely 120.

Initially, the normalized features of the DBs are presented in parallel coordinates plots, figure 6.1. It's highlighted that the smaller DB is a subpart of the larger one.

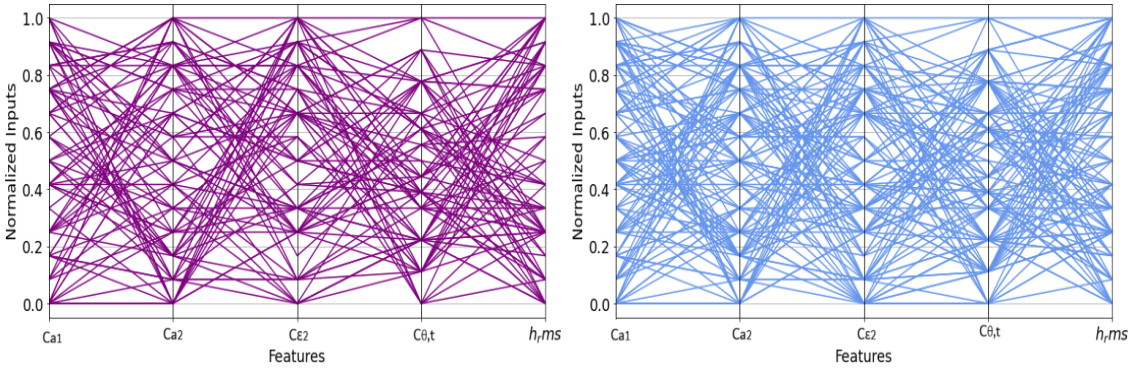

Figure 6.1: *Isolated Wing Case. Parallel coordinate plots for: DB 80 (Left) and DB 120 (Right). As in case I, the effectiveness of the LHS method in covering the input space is noticeable.*

Next, the statistical moments of the C_f for both suction and pressure sides are computed with the rPCE-CFD and illustrated in figures 6.2 and 6.3. In these plots, one may observe the average transition onset, alongside how sensitive it is to the uncertainties under consideration.

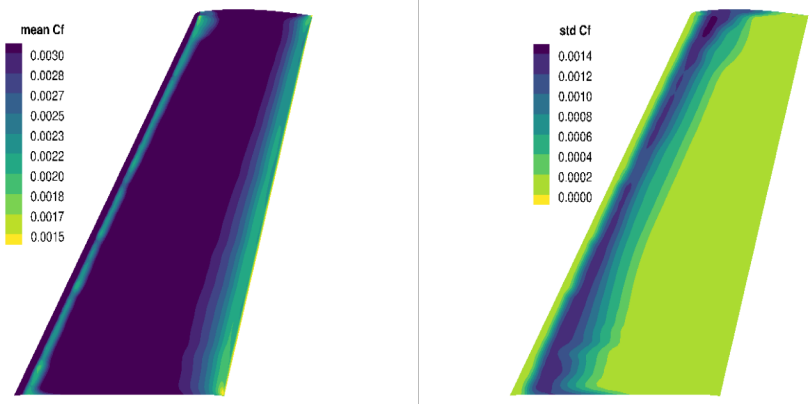

Figure 6.2: *Isolated Wing Case. Statistical moments of C_f, pressure side. Mean (Left) and standard deviation (Right).*

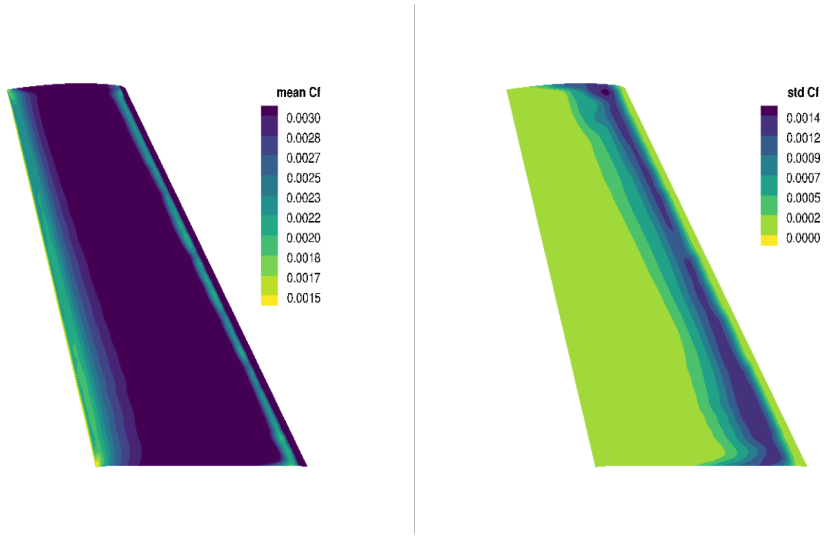

Figure 6.3: *Isolated Wing Case. Statistical moments of C_f, suction side. Mean (Left) and standard deviation (Right).*

C_f is distributed similarly in both the suction and pressure sides. The flow is turbulent in the beginning; then, it becomes laminar for a very thin zone near the leading edge, followed by a turbulent flow region that covers most of the wing's surface. The standard deviation of C_f indicates that the first transition point near the leading edge is sensitive to the uncertainties involved. In particular, it can "move" inside the darker zone near the leading edge, where C_f's standard deviation displays the highest values.

The PCC is produced and illustrated in figure 6.4. The importance of each input seems more balanced than in case I.

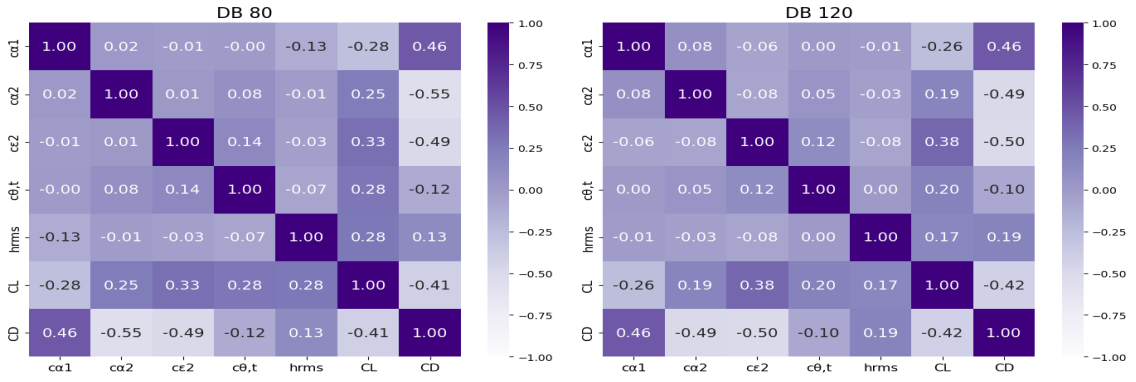

Figure 6.4: *Isolated Wing Case. By observing the C_L, one can see that all input features pose, more or less, correlations of the same strength. Regarding the C_D, it is linearly related mainly to the first three uncertain variables (c_{a1}, c_{a2}, $c_{\varepsilon 2}$) in an equal manner.*

6.2 UQ with DNN

The high-fidelity UQ results are obtained from rPCE-CFD(80) and rPCE-CFD(120). It was decided to train models on both DB-80 and DB-120. For each DB, 90% of the data is used for training and 10% for validation. Every DNN is employed for UQ, and the results are compared with the high-fidelity UQ results. In particular, MC-DNN is conducted for 20000 and 50000 random samples, gPCE-DNN is conducted for k=2 (243 nodes), rPCE-DNN (also k=2) for the 243 GQ nodes as well as for 80 and 120 samples obtained by the LHS.

Two approaches are carried out. Considering the C_D/C_L as the QoI, the first approach involves one single-output DNN, which predicts directly the QoI. The second approach suggests, like in the previous case, the use of two single-output DNNs. One to predict the C_L and the other the C_D. Considering that the two approaches involve 3 models in total, and that the aim is to check the performances of models trained on the two DBs, 6 is the number of models that should be trained and employed for UQ.

Table 6.1: *ONERA M6 Wing Case. UQ with CFD.*

Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}
rPCE-CFD(120)	120	6.8571	0.3278
rPCE-CFD(80)	80	6.8636	0.3180

6.2.1 DNN configuration used in Case I

First, the manually created configuration from case I (table 5.2) is assessed for the creation of the 6 DNNs. For each of DB-80 and DB-120, there are three models: one for C_L, one for C_D, and one for C_L and C_D. Each of these models is used for UQ and the results can be seen in table 6.2.

Table 6.2: *ONERA M6 Wing Case. DNN configuration from case I.*

		1 model		**2 models**	
Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}	μ_{C_D/C_L}	σ_{C_D/C_L}
MC-DNN(50^4)	80	6.95092	0.3762	6.9012	0.3215
MC-DNN(20^4)	80	6.9524	0.3737	6.9029	0.3187
gPCE-DNN(243)	80	6.9543	0.3673	6.9005	0.3222
rPCE-DNN(243)	80	6.9384	0.3350	6.9079	0.3174
rPCE-DNN(120)	80	6.9209	0.2908	6.9199	0.2856
rPCE-DNN(80)	80	6.8838	0.2869	6.9129	0.3013
MC-DNN(50^4)	120	6.8792	0.43093	6.9023	0.4244
MC-DNN(20^4)	120	6.8803	0.4294	6.9023	0.4230
gPCE-DNN(243)	120	6.8810	0.4234	6.8965	0.4163
rPCE-DNN(243)	120	6.8666	0.3737	6.8934	0.3773
rPCE-DNN(120)	120	6.9121	0.3093	6.8686	0.3279
rPCE-DNN(80)	120	6.8295	0.3234	6.8614	0.3354

6.2.2 Configurations found by EASY

Then, EASY is used to find the "best" configuration for every model, and, as in case I, validation loss is selected, for the hyperparameter optimization, as the quantity to be minimized. The UQ results are presented in table 6.5. Overall, 6 different hyperparameter tunings are carried out, yielding 6 DNN configurations. Each optimization, as in case I, took EASY roughly 6 hours of computational time. The resulting models are presented in table 6.3. The validation split is selected to be equal to 0.1 and all batch sizes are equal to the DB sizes and for loss function, MAE is selected everywhere. All models have 5 inputs and 1 output.

Table 6.3: *ONERA M6 Wing Case. DNN configurations found by EASY*

DNN target-DB	Neurons/Hidden Layer	Activation Function
C_D/C_L-80	(256, 1024, 256, 64)	tanh/Sigmoid
C_D/C_L-120	(32, 1024, 64, 4096, 4096, 256, 512, 32, 512)	GELU/ReLU
C_L-80	(4096, 4096, 512, 1024, 1024, 32, 1024, 32, 32)	tanh/Sigmoid
C_L-120	(32, 512, 32, 4096, 4096)	ReLU/tanh
C_D-80	(32, 2048, 128, 1024, 64, 64)	GELU/tanh
C_D-120	(1024, 32, 256, 512, 32, 4096, 128, 64, 64, 512)	GELU/Sigmoid

Regarding the different DNN configurations, EASY produced a wide variety of them. In addition, 5 of the 6 configurations have tanh or Sigmoid as activations of their output layer, which both have a maximum of 1. The validation losses of the total 12 models are presented in table 6.4. The second column refers to the 6 models created from the 5.2 configuration (case I). In comparison, the third one refers to the 6 models created, each on its own, optimized by EASY configuration.

Table 6.4: *ONERA M6 Wing Case. DNN validation loss comparison*

DNN	5.2 configuration from Case I	Easy-founded configurations
C_D/C_L-80	0.04194	0.01552
C_D/C_L-120	0.04547	0.01026
C_L-80	0.06252	0.01914
C_L-120	0.05560	0.01819
C_D-80	0.01476	0.00171
C_D-120	0.01154	0.00158

Table 6.5: *ONERA M6 Wing Case. DNN configuration found by EASY:*

		1 model		**2 models**	
Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}	μ_{C_D/C_L}	σ_{C_D/C_L}
MC-DNN(50^4)	80	6.9583	0.3838	6.9528	0.4020
MC-DNN(20^4)	80	6.9601	0.3799	6.9543	0.4022
gPCE-DNN(243)	80	6.9488	0.3742	6.9523	0.3908
rPCE-DNN(243)	80	6.9449	0.3386	6.9264	0.3523
rPCE-DNN(120)	80	6.9212	0.2738	6.8803	0.3238
rPCE-DNN(80)	80	6.8954	0.3139	6.8620	0.3100
MC-DNN(50^4)	120	6.9252	0.4419	6.9187	0.3920
MC-DNN(20^4)	120	6.9265	0.4418	6.9180	0.3918
gPCE-DNN(243)	120	6.9208	0.4256	6.9167	0.3879
rPCE-DNN(243)	120	6.8927	0.3684	6.9087	0.3581
rPCE-DNN(120)	120	6.8866	0.3072	6.8950	0.3407
rPCE-DNN(80)	120	6.9477	0.2976	6.8463	0.3447

6.2.3 Conclusions

Despite the excessive hyperparameter optimization, the UQ results of tables 6.4 and 6.5 are not that far apart, with the ones found by EASY to be, rather, slightly better. Despite the similar generalization performance, one may see that in table 6.3 EASY-found models produced significantly smaller validation losses than their counterparts. The latter, possibly highlights that the chosen validation DB is very small and does not represent sufficiently the model's generalization capability. Additionally, the configuration 5.2 that was fine-tuned for a quite different problem, provides models with satisfactory performances, in the present one as well.

Considering rPCE-CFD (80 and 120) as reference, the finest surrogate-UQ results are obtained from rPCE-DNN(80 and 120) in the 2 DNN approach. The latter is valid for training on DB-80 and DB-120, showcasing the superiority of using 2 DNNs instead of 1 in predicting the C_L and C_D. The UQ results conducted with the EASY found DNNs (2 model approach) are presented next to the references, for DB-80 and DB-120, in tables 6.6 and 6.7, respectively.

Table 6.6: *ONERA M6 Wing Case. Aggregated UQ results using DB-80:*

Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}
rPCE-CFD(80)	80	6.8636	0.3180
rPCE-DNN(80)	80	6.8620	0.3100
rPCE-DNN(120)	80	6.8803	0.3238
MC-DNN(50^4)	80	6.9528	0.4020
gPCE-DNN(243)	80	6.9523	0.3908

Table 6.7: *ONERA M6 Wing Case. Aggregated UQ results using DB-120:*

Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}
rPCE-CFD(120)	120	6.8571	0.3278
rPCE-DNN(80)	120	6.8463	0.3447
rPCE-DNN(120)	120	6.8950	0.3407
MC-DNN(50^4)	120	6.9187	0.3920
gPCE-DNN(243)	120	6.9167	0.3879

The rest of the UQ methods conducted with the surrogates deviate noticeably from the rPCE-CFD results. The reason is either the error propagation of the DNN, the error of the rPCE method, or the stochastisity of LHS.

6.3 Predicting capabilities of KNN and SVR

40 out of the 120TP, are used to test the predicting performance of two additional ML models: K-Nearest Neighbors and Support Vector Regression next to the DNNs created by 5.2, the manually-found configuration, as well as the C_L-80 and C_D-80 DNNs, found by EASY. Thus, DB-80 is used to train the models, and DB-40 is used solely for testing them. The models are trained to predict either the C_L or the C_D. They are compared using their MAPE metric, computed on the testing DB. Regarding the training costs, KNNs are SVRs that are simpler models, with training times of sub-5 seconds, while DNNs require sub-1 minute, on the RTX3060 GPU. The KNNs and SVRs are created using Scikit-Learn, python's ML library. Their hyperparameter tuning is carried out by employing GridSearchCV, a Scikit-Learn function that performs the tuning by training and evaluating a machine learning model using different combinations of hyperparameters. The configurations of the

present models can be found in tables 6.8, 6.9 and 6.10.

Table 6.8: *ONERA M6 Wing Case. DNN manually created configuration.*

Neurons	Activation Function
(128, 64, 32, 64, 1)	ReLU/tanh

Table 6.9: *ONERA M6 Wing Case. DNN EASY-founded configurations.*

DNN	Neurons	Activation Function
C_L-80	(4096, 4096, 512, 1024, 1024, 32, 1024, 32, 32)	tanh/Sigmoid
C_D-80	(32, 2048, 128, 1024, 64, 64)	GELU/tanh

Table 6.10: *ONERA M6 Wing Case. Configurations of KNNs and SVRs.*

KNN-CL	**KNN-CD**
k : 2	k : 3
SVR-CL	**SVR-CD**
C : 0.1	C : 10
ϵ : 0.01	ϵ : 0.01
kernel: polynomial	kernel: RBF

where polynomial and RBF kernels are the Scikit's default ones, specifically, for the polynomial kernel, a 3rd-degree polynomial is used, and for the RBF kernel, the Gaussian RBF.

The performance metrics of all models are presented in figure 6.5.

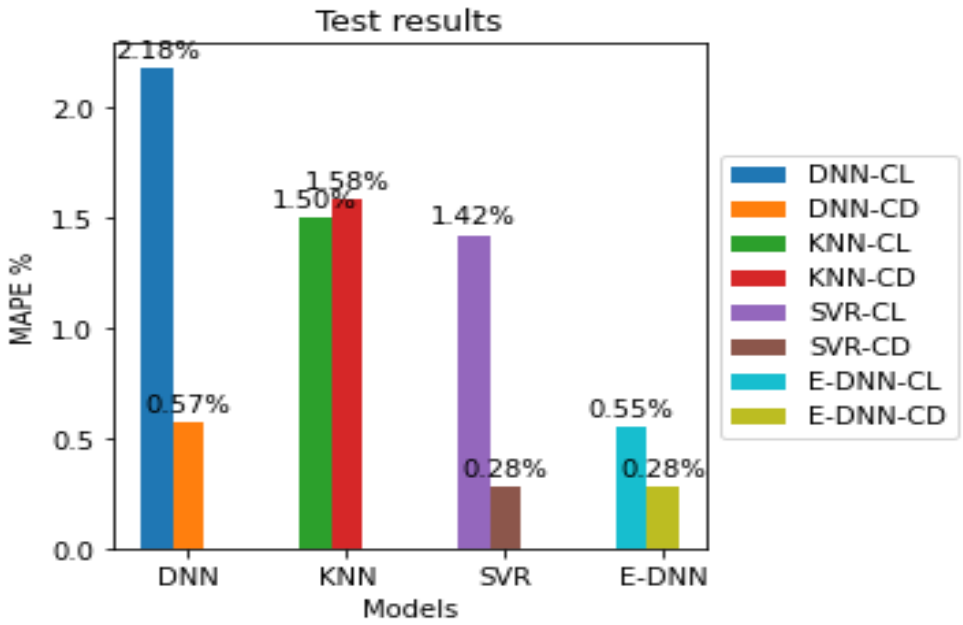

Figure 6.5: *ONERA M6 Wing Case. The EASY-founded models are denoted as "E-DNN". Both KNN and SVR predict the QoIs very satisfactorily. In particular, SVR outperforms the DNN in predicting CL and CD, while KNN-CL outperforms the DNN-CL. E-DNNs are superior and equal to the SVR in predicting CL and CD, respectively.*

The two simpler ML algorithms (KNN and SVR) produce equal and even better-predicting performances than DNNs obtained by the manually created configuration. However, the DNNs found by EASY seem to be superior. It is noted that the Gridsearch function that trained several KNNs and SVRs, aiming to find the best ones, took at most 30 seconds on the RTX3060 GPU.

Chapter 7

Conclusions

7.1 Overview

In this Diploma Thesis, Uncertainty Quantification, the most critical process behind Optimization Under Uncertainties (or Robust Design Optimization) was carried out using three methods: Monte Carlo and two variants of the Non-Intrusive Polynomial Chaos Expansion, gPCE and rPCE. Two aerodynamic problems were studied: the NLF(1)-0416 isolated Airfoil and the ONERA M6 isolated Wing. Since UQ is responsible for the skyrocketed costs of RDO, compared to conventional Optimization, the main objective was the development of Machine Learning models, and mostly DNNs, as surrogates for the expensive Computational Fluid Dynamics code (PUMA software), especially for the task of UQ. All flows were simulated using the RANS solver coupled with Spalart–Allmaras turbulence model and the $\gamma - \tilde{Re}_\theta$ transitional model. UQ was focused on how fluctuations on coefficients that appear in the $\gamma - \tilde{Re}_\theta$ model would impact the transition point, and thus, C_L and C_D. For the training DB creation, Latin Hypercube Sampling was used to obtain samples of the uncertain vectors and PUMA was utilized to evaluate them. The surrogates were trained to predict C_L and C_D, separately or simultaneously, given as inputs the uncertain vectors.

The NLF(1)-0416 was studied for two operating points (F1 and F2), in parallel. An initial DB of 40TP was used to finetune DNN configurations in F1 and afterwards these configurations were trained in 5 different DBs (20, 30, 40, 50, 60 TPs), for F1 and F2. Using these DBs, RBFNs were also trained and both DNNs and RBFNs performance metrics were evaluated in a large unseen DB of 300 samples obtained from the normal distribution. The DNN configurations were found through trial and error manually as well as by employing EASY for hyperameter optimization. The models trained using the configuration found by EASY were the ones who provided

the most accurate test results outperforming the RBFNs for almost every training DB in F1 and F2. Next, UQ was carried out using solely the DNNs trained on the DB-40 and the results were placed next to the ones computed by using the CFD. Then, RBFNs and DNNs trained on every DB were combined through stacking ensemble technique and the resulting model, in most cases, illustrated superior accuracy than the DNN and the RBFN. Last, it is shown that when training a DNN by leaving out the possibly less important input features (feature selection) one can obtain simpler models that are close to the initial one as well as, that it is also possible to end up with a more accurate model.

Regarding the ONERA M6 wing, 120 CFD-evaluated samples were exploited. From this DB, the first 80 TP were extracted, forming a second DB. DNNs were trained on DB-80 and DB-120: a single model to predict C_D/C_L as well as two separate models to predict C_D and C_L individually, and were directly used for UQ. To do so, a well-performing, manually created DNN configuration, from Case I, was employed to create a total of 6 models. Moreover, EASY was set to optimize 6 different configurations, one for each desired model, aiming to obtain 6 better predicting models. It is shown that the two model approach was better than the single model approach for predicting C_D/C_L, and that configurations found by EASY, provided similar results to the manually created configuration. Last, a small study was carried out to explore the ability of KNNs and SVRs to predict the QoI.

7.2 Conclusions

After conducting all the studies above, the following conclusions are drawn:

- Fluctuations in the uncertain variables regarding both applications are observed to shift the transition point considerably, highlighting the severe impact of the uncertainties involved.
- The DNNs could predict C_L and C_D of the assessed transitional flows with sufficient accuracy. As shown in the airfoil case, their use can cut the UQ computational cost more than 50% per candidate solution.
- Using evolutionary algorithms for hyperparameter optimization of the DNNs is a practice that significantly increased the predicting accuracy (up to 82% improvement).
- In the airfoil case, the DNNs proved up to 10 times more accurate than the RBFNs, yet they were more complex and roughly 10 times more expensive to train. Moreover, combining the last two through the stacking ensemble technique provided models that were up to 50% more accurate than the DNNs. The trade-off is that the ensemble model includes the cost of every combined model, which in the present case was trivial.

- Training DNNs using feature selection (airfoil case) yielded simpler models (of up to 20% faster training time) that performed closely to the initial one, and also, one out of the twelve DNNs with reduced inputs, proved 60% more accurate than the initial one in predicting C_D (Test2).
- gPCE proved more accurate than rPCE, which is reasonable since it is subjected to the curse of dimensionality, while rPCE is not. Additionally, as seen in the wing case, when a UQ method requires more calls to the surrogate, its accuracy can decrease due to the enlarged error propagation of the surrogate.
- KNNs and SVRs were also capable of being used as surrogates (wing case), providing results up to 51% better than the ones of DNNs obtained by the manually created DB. Their training was at least 5 times faster than that of DNNs', and they are worth testing before using evolutionary algorithms for DNN hyperparameter tuning.

7.3 Future Work Proposals

Based on the findings of this study, the following future works are proposed:

- ML surrogates can be tested in computationally expensive RDO problems.
- Numerous DNNs and ML models can be developed and stacked through the stacking ensemble. This way, strengths from various models will be leveraged further, and the final accuracy will be the highest possible. Moreover, research can be carried out by testing different models for the role of the meta-learner in the stacking ensemble.
- EASY can be exploited for more complex DNN hyperparameter tunings, including more design variables. It is rather certain that despite the additional cost, the model's accuracy will further improve.
- SVR and RBFN can be investigated further. For example, the performance of the RBFN can be checked by adding hidden layers to its structure.

Appendix A

Appendix

A.1 Statistics and Data Transformation

A.1.1 Fundamentals

A population refers to an entire group being studied, while a sample is a subset of that population used for analysis. The **mean**, also known as the expectation or average of a set, is the summation of the set's values divided by the number of values. **Standard deviation** and **variance** can be considered as the averages of the error and squared error of the set's values calculated w.r.t its mean. In table A.1, x_i stands for the datapoint under consideration and N, n for the number of items of the population and the sample, respectively. For simplicity, only symbol (n) will be used from now on.

Table A.1: *Mean and Standard Deviation*

Type of dataset / Parameter	**Population**	**Sample**
Mean	$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$	$\overline{X} = \frac{\sum_{i=1}^{n} x_i}{n}$
Standard Deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}}$	$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$

Given the above, a new quantity is introduced, which describes the association between two variables (if there is any) in a simple way. Similarly to the definition

of the variance (the average of a single variable's variations from the mean), the **covariance** represents how variations from the mean in the first variable affect the variations from the mean in the second variable. Assuming y_i and $\overline{y}$ the observation and the mean of the second variable the covariance is given by the formula A.1.

$$cov_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1} \tag{A.1}$$

A.1.2 Transforming Data with Normalization and Standardization

Scaling or non-dimensionalization is a standard procedure in many fields, such as fluid dynamics, used to reduce complexity and create more general solutions that can apply to groups of related problems. In this thesis, two scaling techniques were involved: normalization and standardization.

Normalization is the transformation of a dataset to fit into the [0,1] interval while keeping its inherent structure identical to the one before scaling. This procedure requires the formula displayed in table A.2 to be applied separately to each feature. In the context of the present ML algorithms, normalization, if skipped, features with larger magnitudes will disproportionately influence the training phase and lead to bad-performing models. In the scripting environment there are scaling models that can be fitted into the data for this particular task. The so-called scalers learn the extrema of each feature of the training DB and are used to scale new data or unscale an ML model's predictions back to their corresponding magnitudes, using information derived from the initially available data.

Passing on to the second scaling technique, **standardization**. As stated above, the standard deviation measures the average deviation from the mean. We select a random value (x_i) out of a sample. If any distance from the mean ($x_i - \mu$) is divided by the standard deviation, the result will be that distance in standard deviation units. Namely, the new standardized value translates to "how many standard deviations from the mean", and its sign dictates the direction.

The critical difference between the aforementioned two techniques is that normalization fits the data inside the interval [0, 1] without transforming their internal analogies, while standardization centers the data around 0 with a standard deviation of 1.

Table A.2: *Scaling formulas*

Normalization	Standardization
$x_{\mathrm{n}} = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$	$x_{\mathrm{s}} = \frac{x_i - \mu}{\sigma}$

A.1.3 The Pearson Corellation Coefficient

The Pearson product-moment correlation coefficient, or PCC with the symbol r, measures the strength of a relationship between two variables [57]. It is defined as their standardized covariance. It can be seen in A.2 that to standardize the cov_{xy}, one has to divide it by the multiplication of the variable's standard deviations s_x and s_y.

$$r = \frac{cov_{xy}}{s_x s_y} \tag{A.2}$$

It can take values that fall inside the interval [-1,+1], where -1 indicates a perfect negative relationship and +1 a perfect positive one. A negative relationship occurs when one variable increases and the other decreases. The opposite is valid for a positive relationship. The strength of a relationship increases as the absolute value of r increases. A r=0 displays a situation with no relationship between the two variables. In Figure A.1, some different cases are illustrated to provide a better understanding of PCC.

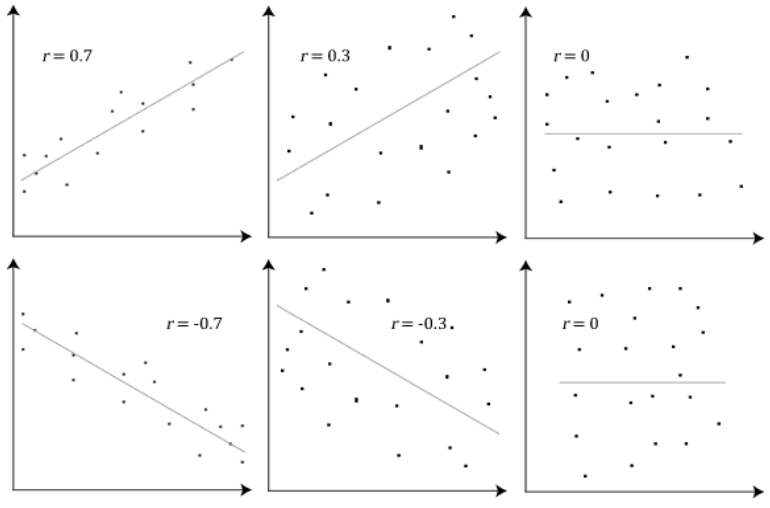

Figure A.1: *PCC example. Figure from [58].*

However, some assumptions - cons accompany the use of PCC. First, PCC is sensitive to outliers, and the correlations that can be detected are only linear ones. Moreover, the variables must be continuous and measured at an interval (must be

evenly spaced) or ratio level (must be evenly spaced and have a natural 0). Last, the data from both variables should follow normal distributions.

A.2 Backpropagation

Backpropagation is the core algorithm behind ANNs. It is performed after every update on the weights and biases of the ANN, and its task is to compute the gradients of the cost function (C) with respect to the weights and biases. Aiming to present how Backpropagation works, it will be explained on the simplest neural network, consided of layers with only one neuron (figure A.2).

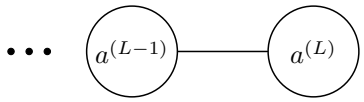

Figure A.2: *Simplest neural network with layers consisted of one neuron*

The cost function $C(w_1, b_1..., w_L, b_L)$, for the k_{th} Training Pattern's target (y_k) is written equal to $\left(a^{(L)} - y_k\right)^2$, for simplicity (eq. A.5). Eqs. A.3 and A.4 illustrate the summation, on the L_{th} layer-neuron, before and after the application of the activation function (σ), respectively.

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)} \tag{A.3}$$

$$a^{(L)} = \sigma(z^{(L)}) \tag{A.4}$$

$$C_0 = \left(a^{(L)} - y_0\right)^2 \tag{A.5}$$

As stated earlier, the goal of Backpropagation is the computation of the gradients of the cost function with respect on all weights and biases.

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

The eqs. A.3, A.4 and A.5 illustrate how the weights and biases influence the cost function, and thus, the chain rule is applied as follows:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} \tag{A.6}$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} \tag{A.7}$$

The terms of eqs. A.6 and A.7 read:

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y_0) \tag{A.8}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}) \tag{A.9}$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \tag{A.10}$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1 \tag{A.11}$$

It is highlighted that since this calculation is solely for the 0_{th} TP, one must take the averages of all TP:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{k} \sum_{i=0}^{k-1} \frac{\partial C_i}{\partial w^{(L)}} \tag{A.12}$$

and

$$\frac{\partial C}{\partial b^{(L)}} = \frac{1}{k} \sum_{i=0}^{k-1} \frac{\partial C_i}{\partial b^{(L)}} \tag{A.13}$$

Eqs. A.12 and A.13 present the calculation of the last two components of the gradient vector. At this point, one is able to compute the gradient of the cost function with respect to $a^{(L-1)}$, and "here is the idea of propagating backwards comes in" as said in [59]:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} \tag{A.14}$$

where from eq. A.3 there is

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)} \tag{A.15}$$

Now by iterating backwards this chain rule method, all components of the gradient vector can be computed.
The arising question at this point is what happens when the ANN has mutliple neurons on every layer and not just one.
In an ANN where every layer has multiple neurons, every layer must have its own neuron counter, an additional symbol (regarding every layer) that will be inserted to the equations as subscript. Assuming that layer L neurons are indexed by the letter n, and the ones of layer (L-1) are indexed by the letter m.

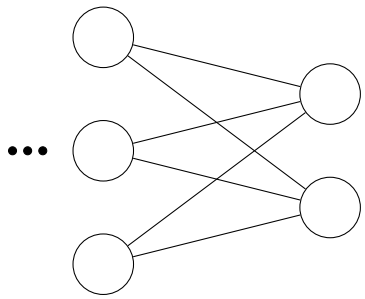

Figure A.3: *Two-layer neural network with 3 neurons in the first layer and 2 neurons in the output layer.*

The cost function this time is the average of the errors of the output neurons, but for simplicity only their sum is considered:

$$C_0 = \sum_{n=0}^{1} \left(a_n^{(L)} - y_{0n}\right)^2 \tag{A.16}$$

Eqs. A.6 and A.7 will stay practically the same, by only adding the appropriate subscripts:

$$\frac{\partial C_0}{\partial w_{mn}^{(L)}} = \frac{\partial z_n^{(L)}}{\partial w_{mn}^{(L)}} \cdot \frac{\partial a_n^{(L)}}{\partial z_n^{(L)}} \cdot \frac{\partial C_0}{\partial a_n^{(L)}} \tag{A.17}$$

$$\frac{\partial C_0}{\partial b_{mn}^{(L)}} = \frac{\partial z_n^{(L)}}{\partial b_{mn}^{(L)}} \cdot \frac{\partial a_n^{(L)}}{\partial z_n^{(L)}} \cdot \frac{\partial C_0}{\partial a_n^{(L)}} \tag{A.18}$$

However, this time each neuron in layer (L-1) affects the cost function through several distinct pathways. Thus, the gradient of the cost function w.r.t. $a_n^{(L)}$ (eq. A.14) must change. Essentially, the influences of $a_n^{(L-1)}$ to the cost function must be summed over layer L.

$$\frac{\partial C_0}{\partial a_m^{(L-1)}} = \sum_{n=0}^{1} \frac{\partial z_n^{(L)}}{\partial a_m^{(L-1)}} \cdot \frac{\partial a_n^{(L)}}{\partial z_n^{(L)}} \cdot \frac{\partial C_0}{\partial a_n^{(L)}} \tag{A.19}$$

A.3 Radial Basis Networks

The general concept of Radial Basis Function Networks is that they are simple three-layer feedforward fully connected neural networks in which the role of the activation function is held by a specific group of functions known as Radial Basis. This type of network is commonly used for function approximation in regression tasks. It is highlighted that RBFNs assessed in this work are not trained using backpropagation. The approach followed for this type of network can be found at [60].

The Gaussian RBF

RBF is a function whose output is highly influenced by the Euclidean distance between some input x and a fixed center c_i, the center of the RBF. This is how its name came about. There are many types of RBF functions, out of which the most known and utilized in this thesis is the Gaussian RBF

$$\Phi_i = e^{-\frac{(x-c_i)^2}{\sigma_i^2}}$$

where σ_i is the variance around the center, also known as radius, which can be considered as a hyperparameter of the model when training an RBFN.

Bibliography

[1] K. Giannakoglou. Optimization methods. Lectures, School of Mechanical Engineering, NTUA, 2022.

[2] K. Giannakoglou. Optimization methods in aerodynamics, 2006.

[3] M. Cavazzuti. *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics*, pages 77–102. Springer Berlin Heidelberg, 2013.

[4] Easy - the evolutionary algorithms system home. `http://velos0.ltt.mech.ntua.gr/EASY/`.

[5] G. Pampalis. Implementation of polynomial chaos expansion in aerodynamic robust design - optimization with evolutionary algorithms under stochastic inputs. Diploma thesis, NTUA, 2015. Supervised by Prof. Kyriakos Giannakoglou.

[6] H. Beyer and B. Sendhoff. Robust optimization – a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33):3190–3218, 2007.

[7] M. Vasile. *Optimization Under Uncertainty with Applications to Aerospace Engineering*. Springer, 2021. Chapter 13.

[8] B. El-Haik and R. Al-Aomar. *Simulation-based Lean Six-Sigma and Design for Six-Sigma*. John Wiley & Sons, 2006.

[9] K. Shimoyama, A. Oyama, and K. Fujii. Development of multi-objective six sigma approach for robust design optimization. *Journal of Aerospace Computing, Information, and Communication*, 5(8):215–233, 2008.

[10] Y. Nomaguchi, K. Fujita, Y. Kishita, M. Uwasu, et al. Robust design of system of systems using uncertainty assessment based on lattice point approach: Case study of distributed generation system design in a japanese dormitory town. *International Journal of Automation Technology*, September 2016.

[11] K. Fragkos, E. Papoutsis-Kiachagias, and K. Giannakoglou. Pfosm: An efficient algorithm for aerodynamic robust design based on continuous adjoint and matrix-vector products. *Computers & Fluids*, 2019.

[12] N. Morozova, F. Trias, R. Capdevila, E. Schillaci, and A. Oliva. A CFD-based surrogate model for predicting flow parameters in a ventilated room using sensor readings. *Energy and Buildings*, 2022.

[13] T. Zhang, B. Dey, K. Veeraraghavan, H. Kulkarni, and A. Chakraborty. Demystifying the data need of ML-surrogates for CFD simulations. *arXiv preprint arXiv:2205.08355*, 2022.

[14] M. Martinelli and D. Duvigneau. Comparison of second-order derivatives and metamodel-based monte-carlo approaches to estimate statistics for robust design of a transonic wing. In *Proceedings of the 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2008.

[15] Z. Song, Z. Liu, J. Lu, and C. Yan. Quantification of parametric uncertainty in $\gamma - \tilde{Re}_\theta$ model for typical flat plate and airfoil transitional flows. *Chinese Journal of Aeronautics*, 2023.

[16] K. Duraisamy, Z. Zhang, and A. P. Singh. New approaches in turbulence and transition modeling using data-driven techniques. In *Proceedings of the 53rd AIAA Aerospace Sciences Meeting*, 2015.

[17] Y. Liu, D. Wang, X. Sun, Y. Liu, N. Dinh, and R. Hu. Uncertainty quantification for multiphase-CFD simulations of bubbly flows: a machine learning-based bayesian approach supported by high-resolution experiments. *Reliability Engineering & System Safety*, 2021.

[18] OpenAI. Introducing chatgpt. `https://openai.com/index/chatgpt/`, 2023.

[19] OpenAI et al. Gpt-4 technical report, 2024.

[20] OpenAI. Sora: Creating video from text. Technical report, 2024.

[21] Andreas B., Tim D., Sumith K., Daniel M., Maciej K., Dominik L., Yam L., Zion E., Vikram V., Adam L., Varun J., and Robin R. Stable video diffusion: Scaling latent video diffusion models to large datasets, 2023.

[22] L. Yixin, Z. Kai, L. Yuan, Y. Zhiling, G. Chujie, C. Ruoxi, Y. Zhengqing, H. Yue, S. Hanchi, G. Jianfeng, H. Lifang, and Lichao S. Sora: A review on background, technology, limitations, and opportunities of large vision models, 2024.

[23] X. Tang. The role of artificial intelligence in medical imaging research. *BJR—Open*, 2020.

[24] A. Amini and A. Soleimany. 6.s191: Introduction to deep learning, 2023.

[25] A. Balodi. Application of introduction artificial intelligence machine learning in real life, 04 2020.

[26] J. Peng, E. Jury, P. Dönnes, and C. Ciurtin. Machine learning techniques for personalised medicine approaches in immune-mediated chronic inflammatory

diseases: applications and challenges. *Frontiers in pharmacology*, 12:720694, 2021.

[27] Overfitting. https://www.mathworks.com/discovery/overfitting.html.

[28] Simon Fraser University. Chapter 2: Ordinary least squares. https://www.sfu.ca/~dsignori/buec333/lecture%208.pdf.

[29] P. Rishith. A simple approach to classification and regression. https://pub.towardsai.net/understanding-k-nearest-neighbors-a-simple-approach-to-classification-and-regression-e4b30b37f151.

[30] A. Moradzadeh, A. Mansour-Saatloo, B. Mohammadi-Ivatloo, and A. Anvari-Moghaddam. Performance evaluation of two machine learning techniques in heating and cooling loads forecasting of residential buildings. *Applied Sciences*, 10(11), 2020.

[31] C. Yan, Z. Yin, X. Shen, D. Mi, F. Guo, and D. Long. Surrogate-based optimization with improved support vector regression for non-circular vent hole on aero-engine turbine disk. *Aerospace Science and Technology*, 96:105332, 2020.

[32] JacobSoft. Support vector regression (svr). https://www.jacobsoft.com.mx/en/support-vector-regression/.

[33] Deep Learning (DL) — s.mriquestions.com. https://s.mriquestions.com/what-is-a-neural-network.html#.

[34] D. Abueidda, Q. Lu, and S. Koric. Deep learning collocation method for solid mechanics: Linear elasticity, hyperelasticity, and plasticity as examples, 12 2020.

[35] Understanding the universal approximation theorem. https://medium.com/@ML-STATS/understanding-the-universal-approximation-theorem-8bd55c619e30.

[36] X. Qi, J. Wang, Y. Chen, Y. Shi, and L. Zhang. Lipsformer: Introducing lipschitz continuity to vision transformers, 04 2023.

[37] Papers with Code. Xavier initialization explained. https://paperswithcode.com/method/xavier-initialization.

[38] R. Zaheer and H. Shaziya. A study of the optimization algorithms in deep learning. In *2019 Third International Conference on Inventive Systems and Control (ICISC)*, pages 536–539, 2019.

[39] A. Azri, C. Favre, N. Harbi, J. Darmont, and C. Noûs. Rumor classification through a multimodal fusion framework and ensemble learning. *Information Systems Frontiers*, pages 1 – 16, 2022.

[40] V. Asouti, Xenofon Trompoukis, I. Kampolis, and K. Giannakoglou. Unsteady cfd computations using vertex-centered finite volumes for unstructured grids

on graphics processing units. *International Journal for Numerical Methods in Fluids*, 67:232 – 246, 09 2011.

[41] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

[42] M. Piotrowski and D. Zingg. Smooth local correlation-based transition model for the spalart–allmaras turbulence model. *AIAA Journal*, 59(2):474–492, 2021.

[43] T. Sullivan. *Introduction to uncertainty quantification*, volume 63. Springer, 2015.

[44] A. Quarteroni. Mathematical models in science and engineering. *Notices of the AMS*, 2009.

[45] National Research Council, Division on Engineering, Physical Sciences, Board on Mathematical Sciences, Their Applications, Committee on Mathematical Foundations of Verification, and Uncertainty Quantification. *Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification*. National Academies Press, 2012.

[46] T. Dowell. Understanding uncertainty quantification: The different types, 2023. Available at `https://www.digilab.co.uk/news/understanding-uncertainty-quantification-different-types-of-uncertainty` [Accessed March 2023].

[47] A. Owen. *Monte Carlo Theory, Methods and Examples*. Stanford University, 2013.

[48] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[49] Monte carlo method. `https://en.wikipedia.org/wiki/Monte_Carlo_method`.

[50] N. Wiener. The homogeneous chaos. *American Journal of Mathematics*, pages 897–936, 1938.

[51] Notepub's official team. Descriptive statistics - raw and central moments, Aug 2021.

[52] E. Papoutsis-Kiachagias, V. Asouti, and K. Giannakoglou. Assessment of variants of the method of moments and polynomial chaos approaches to aerodynamic uncertainty quantification. In *4th International Conference on Uncertainty Quantification in Computational Sciences and Engineering*, Athens, 2021. Institute of Research and Development for Computational Methods in Engineering Sciences (ICMES).

[53] C. Blum, U. Steinseifer, and M. Neidlin. Systematic analysis of non-intrusive polynomial chaos expansion to determine rotary blood pump performance over the entire operating range. *Computers in Biology and Medicine*, 168:107772, 2024.

[54] M. Kontou V. Asouti and K. Giannakoglou. Rbf surrogates for uncertainty quantification and aerodynamic shape optimization under uncertainties. *MPDI*, 2023.

[55] AIAA Transition Modeling Workshop-I — Transition Modeling and CFD Vision 2030 — transitionmodeling.larc.nasa.gov. `https://transitionmodeling.larc.nasa.gov/workshop_i/?doing_wp_cron=1719498266.2547850608825683593750`.

[56] D. Somers. Design and experimental results for a natural-laminar-flow airfoil for general aviation applications. Technical report, NASA, 1981.

[57] A. Field. *Discovering Statistics Using SPSS*. SAGE, London, 2009.

[58] Pearson product-moment correlation. `https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php`.

[59] 3Blue1Brown. Backpropagation calculus — chapter 4, deep learning. YouTube Video, 2017. `https://www.youtube.com/watch?v=tIeHLnjs5U8`.

[60] K. C. Giannakoglou, V. Asouti, and D. Kapsoulis. Low-cost optimization using evolutionary algorithms for engineering applications, 2023.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Υποκατάστατα Μοντέλα βασισμένα στη Μηχανική Μάθηση για Ποσοτικοποίηση Αβεβαιοτήτων στην Υπολογιστική Ρευστοδυναμική

Διπλωματική Εργασία - Εκτενής Περίληψη στα Ελληνικά

Διονύσιος Μπακής

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Εισαγωγή

Η Διπλωματική αυτή Εργασία υλοποιείται στην περιοχή της βελτιστοποίησης μορφής σωμάτων με αεροδυναμικά κριτήρια, λαμβάνοντας υπόψη αβεβαιότητες. Ειδικότερα, διερευνά την επάρκεια των πλήρως συνδεδεμένων βαθιών νευρωνικών δικτύων πρόσθιας τροφοδότησης ως υποκατάστατων του υψηλής πιστότητας αλλά πολύ ακριβού, επιλύτη Υπολογιστικής Ρευστοδυναμικής για την Ποσοτικοποίηση Αβεβαιότητας (UQ). Αυτή είναι η βασική διαδικασία στην αναζήτηση της βέλτιστης αεροδυναμικής μορφής που μπορεί να θεωρηθεί λιγότερο ευαίσθητη στην επίδραση αβεβαιοτήτων (Στιβαρός Σχεδιασμός ή UQ).

Η UQ πραγματοποιείται με τη χρήση Monte Carlo (MC) και της μεθόδου του μη-επεμβατικού αναπτύγματος του πολυωνυμικού χάους (niPCE) σε συνδυασμό με τον επιλύτη Υπολογιστικής Ρευστοδυναμικής (CFD) ή ένα υποκατάστατο μοντέλο για δύο αεροδυναμικά προβλήματα που αφορούν μεταβατικές ροές: την NLF(1)-0416 μεμονωμένη αεροτομή και την ONERA M6 μεμονωμένη πτέρυγα. Οι ροές προσωμοιώνονται με το οικείο λογισμικό PUMA της Μονάδας Παράλληλης Ρευστοδυναμικής & Βελτιστοποίησης του ΕΜΠ, επιλύοντας τις Reynolds-Averaged Navier-Stokes εξισώσεις μαζί με το Spalart-Allmaras μοντέλο τύρβης και το μοντέλο μετάβασης $\gamma - \tilde{R}e_\theta$. Μελετώνται περιπτώσεις με αβεβαιότητες που σχετίζονται με συντελεστές που εμφανίζονται στο μοντέλο μετάβασης $\gamma - \tilde{R}e_\theta$.

Βελτιστοποίηση υπό Αβεβαιότητες

Σχεδόν όλα τα προβλήματα του πραγματικού κόσμου, όπως η βελτιστοποίηση σχήματος, υπόκεινται σε κάποιο βαθμό αβεβαιότητας. Μπορεί να σχετίζεται με στοχαστικές διαταραχές που επηρεάζουν το περιβάλλον, τις παραμέτρους σχεδιασμού ή την αξιολόγηση του συστήματος.

Ως αποτέλεσμα, εισάγεται η «Βελτιστοποίηση υπό Αβεβαιότητες» ή «Στιβαρός Σχεδιασμός». Στο πλαίσιο αυτής της εργασίας, η έμφαση δίνεται στις αβεβαιότητες που σχετίζονται με τις περιβαλλοντικές παραμέτρους του συστήματος. Έτσι, πραγματοποιείται μια ταξινόμηση μεταξύ των μεταβλητών εισόδου του συστήματος. Καταρχάς, υπάρχουν οι N μεταβλητές σχεδιασμού ($\vec{b} \in R^N$), οι οποίες βρίσκονται υπό τον έλεγχο του σχεδιαστή. Υπάρχουν επίσης οι M περιβαλλοντικές, στιβαρές ή αβέβαιες μεταβλητές ($\vec{c} \in R^M$), οι οποίες είναι παράμετροι που αποτελούν μέρος του περιβάλλοντος του συστήματος αλλά υπόκεινται σε κάποιο βαθμό στοχαστικότητας. Οι περιβαλλοντικές μεταβλητές είναι εκτός του ελέγχου του σχεδιαστή.

Για τη μοντελοποίηση των αβέβαιων μεταβλητών, ο χρήστης πρέπει να υποθέσει ότι κάθε μία από αυτές ακολουθεί μια γνωστή συνάρτηση πυκνότητας πιθανότητας (PDF). Στην παρούσα μελέτη, όλες οι αβέβαιες μεταβλητές θεωρήθηκαν κανονικά κατανεμημένες γύρω από τις μέσες τιμές τους εντός του διαστήματος $[\mu_i - 3\sigma_i, \mu_i + 3\sigma_i]$, (τρία σίγμα).

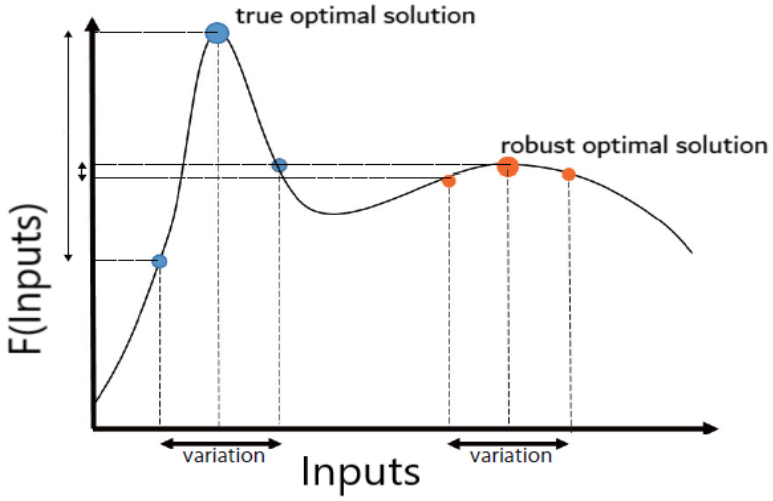

Figure 7.4: *Καθολική βέλτιστη έναντι στιβαρής βέλτιστης λύσης. Εάν ο σχεδιασμός γίνεται για την πραγματική βέλτιστη λύση, δηλαδή το καθολικό μέγιστο της συνάρτησης, μικρές αβεβαιότητες στις μεταβλητές εισόδου μπορούν να οδηγήσουν σε εξαιρετικά ανεπιθύμητα αποτελέσματα. Ως εκ τούτου, ο εντοπισμός της στιβαρής βέλτιστης λύσης σημαίνει την εύρεση μιας περιοχής που μοιάζει με κοιλάδα εντός του πεδίου τιμών. Στην περίπτωση αυτή, οι μεταβολές στις εισόδους οδηγούν σε αμελητέες μεταβολές στην τιμή εξόδου της F, καθιστώντας τον σχεδιασμό πολύ πιο αξιόπιστο. Σχήμα από [10]*

Μηχανική Μάθηση και Βαθιά Νευρωνικά δίκτυα

Η Μηχανική Μάθηση (ML) και το υποπεδίο της, η Βαθιά Μάθηση (DL), είναι δύο ολοένα και πιο δημοφιλείς τομείς όπου αλγόριθμοι μπορούν να εκπαιδευτούν σε έναν συγκεκριμένο όγκο δεδομένων και στη συνέχεια να εκτελέσουν προβλέψεις σε νέα αθέατα δεδομένα. Όσον αφορά τα υποκατάστατα μοντέλα για CFD, οι αλγόριθμοι που χρησιμοποιούνται για την, χαμηλού κόστους, προσέγγιση συναρτήσεων ανήκουν στην κατηγορία ML που είναι γνωστή ως μάθηση με επίβλεψη (supervised learning). Ένας επιλεγμένος αριθμός διανυσμάτων εισόδου αξιολογείται με τη χρήση του CFD και τα ζεύγη εισόδου-εξόδου στοιβάζονται, σχηματίζοντας τη λεγόμενη Βάση Δεδομένων (DB) εκπαίδευσης. Τα Τεχνητά Νευρωνικά Δίκτυα (ANNs), που απεικονίζονται στο σχήμα 7.5, είναι ο κύριος τύπος μοντέλου που διερευνάται στην παρούσα εργασία. Είναι υπολογιστικά συστήματα σχεδιασμένα να μιμούνται τη λειτουργία του ανθρώπινου εγκεφάλου και θεωρούνται μέρος του ευρύτερου πεδίου της μηχανικής μάθησης. Βρίσκονται ακριβώς στη διαχωριστική γραμμή μεταξύ ML και DL. Αυτό ισχύει διότι, με απλά λόγια, τα Βαθιά Νευρωνικά Δίκτυα (DNNs) είναι ANNs με περισσότερα από ένα κρυφά στρώματα. Υπάρχουν δύο κατηγορίες παραμέτρων που εμπλέκονται στα ANNs. Πρώτον, οι δύο τύποι εκπαιδεύσιμων παραμέτρων, δηλαδή τα βάρη και οι όροι bias, των οποίων οι τιμές προσαρμόζονται κατά τη διάρκεια της εκπαίδευσης. Δεύτερον, οι πολυάριθμες υπερπαραμέτροι, όπως ο αριθμός των στρωμάτων, οι νευρώνες ανά στρώμα και οι τύποι συναρτήσεων ενεργοποίησης, οι οποίες είναι παράμετροι που καθορίζονται από τον χρήστη πριν από την έναρξη της

εκπαίδευσης.

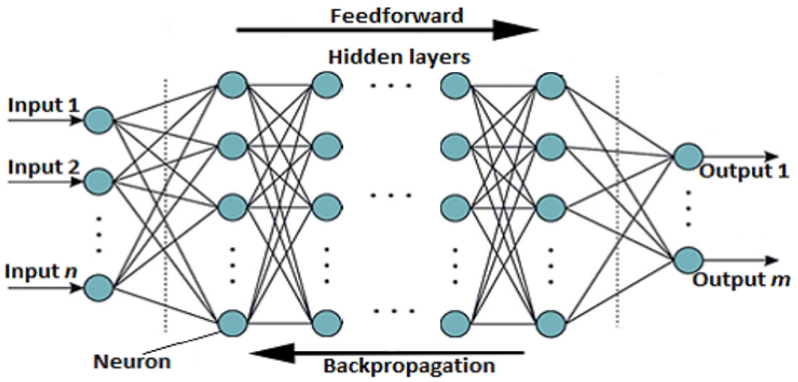

Figure 7.5: *Πλήρως συνδεδεμένο DNN πρόσθιας τροφοδότησης. Σχήμα από [34]*

Κάθε νευρώνας, εκτός από εκείνους του στρώματος εισόδου, λαμβάνει πληροφορίες από τους νευρώνες του προηγούμενου στρώματος και στη συνέχεια μεταβιβάζει τις επεξεργασμένες, μέσω της εξίσωσης 7.20, πληροφορίες στους νευρώνες του επόμενου στρώματος. Ο τύπος υπολογισμού της εξόδου του νευρώνα έχει ως εξής:

$$y = f(\sum x_i w_i + b) \tag{7.20}$$

όπου y είναι η έξοδος του νευρώνα, $\sum x_i w_i$ είναι το άθροισμα των εξόδων κάθε νευρώνα του προηγούμενου επιπέδου πολλαπλασιασμένες με τα αντίστοιχα βάρη, b είναι ο όρος bias και f είναι η μη γραμμική συνάρτηση ενεργοποίησης.

Μέθοδοι Ποσοτικοποίησης Αβεβαιότητας

Η μέθοδος Monte Carlo είναι μια υπολογιστική τεχνική που χρησιμοποιείται για την εκτίμηση των πιθανών αποτελεσμάτων ενός αβέβαιου γεγονότος ή για την προσέγγιση λύσεων σε μαθηματικά προβλήματα που περιλαμβάνουν τυχαίες μεταβλητές. Υποθέτοντας ένα πρόβλημα με M αβέβαιες μεταβλητές, η χρήση της MC για UQ έχει ως εξής. Αρχικά, ένας επιθυμητός αριθμός τυχαίων διανυσμάτων (N) πρέπει να αντληθεί από την κοινή κατανομή τους $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_M$. Για να γίνει αυτό, λαμβάνονται N τυχαίοι αριθμοί από κάθε κατανομή $\mathcal{D}_i$, δημιουργώντας τα αβέβαια δειγματικά διανύσματα. Στη συνέχεια, αφού κάποιο μοντέλο υπολογίσει τις εξόδους τους, εξάγονται οι στατιστικές ροπές των Ποσοτήτων Ενδιαφέροντος (QoIs).

Η μέθοδος niPCE αφορά την ορθογώνια διάσπαση μιας στοχαστικής QoI σε μια κατάλληλη σειρά, με στόχο τον αναλυτικό προσδιορισμό των στατιστικών ροπών του αποκομμένου αναπτύγματος. Ο μέγιστος βαθμός του αναπτύγματος είναι γνωστός ως «τάξη χάους» (k). Στο niPCE, η αντικειμενική συνάρτηση $F(\vec{c})$ αντιμετωπίζεται ως «μαύρο κουτί» και αναπτύσσεται ως γραμμικός συνδυασμός μιας οικογένειας

πολυδιάστατων ορθοκανονικών πολυωνύμων ως εξής:

$$F(\vec{c}) \approx \sum_{i=0}^{Q-1} J_i P_i(\vec{c}) \tag{7.21}$$

όπου $Q = \frac{(M+k)!}{M!k!}$ είναι ο μεγαλύτερος βαθμός των ορθοκανονικών πολυδιάστατων πολυωνύμων $P_i(\vec{c})$, και J_i είναι τα αντίστοιχα βάρη τους.

Ο μέσος όρος και η τυπική απόκλιση υπολογίζονται ως εξής:

$$\mu_F = J_0, \quad \sigma_F = \sqrt{\sum_{i=1}^{Q-1} J_i^2} \tag{7.22}$$

Ο υπολογισμός του J_i πραγματοποιείται είτε με χρήση Gauss Hermite Quadrature είτε με χρήση μιας προσέγγισης που βασίζεται στη γραμμική παλινδρόμηση, οπότε προκύπτουν οι δύο παραλλαγές του niPCE, που αξιοποιήθηκαν: Gauss Quadrature PCE (gPCE) και Regression PCE (rPCE).

Ποσοτικοποίηση Αβεβαιότητας στην μεμονωμένη αεροτομή NLF(1)-0416

Αξιολογείται η χρήση των DNN για UQ στην περίπτωση ενός προβλήματος ροής γύρω από μια μεμονωμένη αεροτομή. Τα DNN συγκρίνονται με το CFD και το υποκατάστατο RBFN που χρησιμοποιείται στο [54] για τον ίδιο σκοπό. Εξετάζονται τέσσερις αβεβαιότητες που σχετίζονται με τέσσερις (M=4) σταθερές ($c_{a1}, c_{a2}, c_{\varepsilon 2}, c_{\theta,t}$) του μοντέλου μετάβασης $\gamma - \tilde{Re}_\theta$. Οι τελευταίες επιλέχθηκαν ως πηγές αβεβαιότητας του μοντέλου, καθώς βαθμονομήθηκαν με βάση εμπειρικές πληροφορίες που προέρχονται από πειράματα. Ως εκ τούτου, είναι εύλογο ότι οι τιμές τους ενδέχεται να μην είναι οι πλέον κατάλληλες για ορισμένες άλλες περιπτώσεις. Στόχος ήταν να ποσοτικοποιηθεί η επίδρασή τους στους συντελεστές Άνωσης και Οπισθέλκουσας (C_L και C_D) της αεροτομής. Συγκεκριμένα, όλες οι αβέβαιες μεταβλητές θεωρήθηκαν κανονικά κατανεμημένες γύρω από τις ονομαστικές τους τιμές (μέσες τιμές) με αυθαίρετα επιλεγμένες (10% των μέσων τιμών τους) τυπικές αποκλίσεις (σχήμα 7.6).

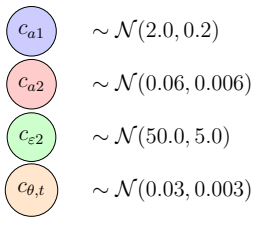

Figure 7.6: *NLF(1)-0416 αεροτομή. Αβέβαιες μεταβλητές*

Τα μοντέλα εκπαιδεύτηκαν ώστε να λαμβάνουν τα αβέβαια διανύσματα μεταβλητών ως είσοδους και να προβλέπουν τις επιθυμητές Ποσότητες Ενδιαφέροντος (QoIs), που διαφορετικά υπολογίζονταν με την εκτέλεση ενός ακριβού κώδικα CFD. Επιπλέον,

χρησιμοποιώντας τα υποκατάστατα, πραγματοποιήθηκε ανάλυση ευαισθησίας για τις τρεις εξεταζόμενες μεθόδους UQ, καθώς αυτό ήταν πολύ φθηνότερο (σχεδόν δωρεάν με ένα εκπαιδευμένο υποκατάστατο) από ό,τι με τον κώδικα CFD.

Χρησιμοποιήθηκε ένα δομημένο πλέγμα τύπου C 705x97 κόμβων με μέγιστο $y^+ = 0,74$, σχηματιζόμενο από τετράπλευρα, το οποίο βρέθηκε στο [55].

Οι ακόλουθες μελέτες ασχολούνται με την αεροτομή NLF(1)-0416 για δύο διαφορετικές συνθήκες ροής, όπως στον πίνακα 7.3.

Table 7.3: *NLF(1)-0416 αεροτομή. Συνθήκες ροής*

Flows	$\mathbf{M_\infty}$	$\mathbf{a_\infty}$	$\mathbf{Re_c}$
F1	0.1	2.03^o	4×10^6
F2	0.3	4.07^o	6×10^6

Η ένταση της τύρβης της ελεύθερης ροής είναι Tu=0,15% και στις δύο περιπτώσεις.

Με στόχο τη δημιουργία της DB εκπαίδευσης, για κάθε συνθήκη ροής, χρησιμοποιήθηκε η μέθοδος δειγματοληψίας λατινικού υπερκύβου (LHS) για τη δειγματοληψία 40 διανυσμάτων (με 4 καταχωρήσεις το καθένα) από το χώρο των αβέβαιων μεταβλητών. Στη συνέχεια, ο επιλύτης RANS υπολόγισε τη ροή και δημιουργήθηκαν δύο DB εκπαίδευσης.

Στο Σχήμα 7.7 παρουσιάζονται οι κατανομές του συντελεστή τριβής (C_f), οι οποίες απεικονίζονται για τις ονομαστικές τιμές των αβέβαιων μεταβλητών μαζί με αυτές που παράγονται από τα 39 πρόσθετα δείγματα. Το F1 συμβολίζει τη «Ροή 1» και το F2 τη «Ροή 2» Οι απότομες γραμμές σε αυτό το διάγραμμα αντιστοιχούν σε σημεία μετάβασης από στρωτή σε τυρβώδη ροή.

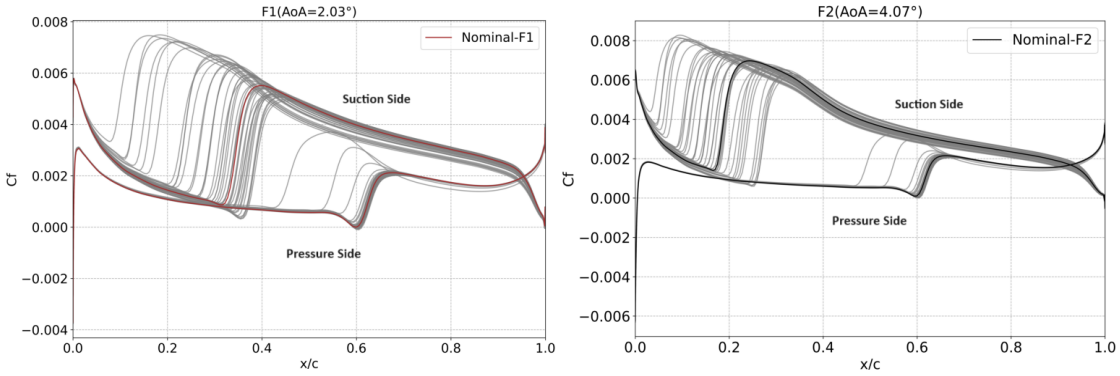

Figure 7.7: *NLF(1)-0416 αεροτομή. Κατανομές C_f και για τις δύο συνθήκες ροής. Μάλλον μικρές αλλαγές στις αβέβαιες μεταβλητές έχουν μεγάλο αντίκτυπο στο σημείο μετάβασης. Στη δεύτερη περίπτωση παρατηρείται μεγαλύτερη τυρβώδης περιοχή γύρω από το τμήμα της αεροτομής. Το τελευταίο συμβαίνει κυρίως λόγω της υψηλότερης γωνίας προσβολής.*

Μαζί με την πρώτη DB των 40 TP, στην οποία τελειοποιήθηκε κάθε μοντέλο αυτού του κεφαλαίου (επιλογή της αρχιτεκτονικής και των υπερπαραμέτρων), δημιουργήθηκαν επίσης τέσσερις ακόμη DBs διαφορετικών μεγεθών με τη χρήση δειγματοληψίας LHS. Αυτό το βήμα έγινε για να ελεγχθούν οι δυνατότητες γενίκευσης των διαμορφώσεων των μοντέλων που βελτιστοποιήθηκαν για μια συγκεκριμένη DB, όταν εκπαιδεύονται σε διαφορετικές (κατά μέγεθος). Η διαδικασία πραγματοποιήθηκε και για τις δύο συνθήκες ροής. Οι πρόσθετες DB ήταν όλες ανεξάρτητες μεταξύ τους (δηλαδή οι μικρότερες δεν αποτελούσαν υποσύνολα των μεγαλύτερων) και αποτελούνταν από 20, 30, 50 και 60 πρότυπα εκπαίδευσης (TP), αντίστοιχα. Υπογραμμίζεται ότι όλα τα μοντέλα τελειοποιήθηκαν στη ΒΔ των 40 TP, αποκλειστικά για την F1, και ότι όλες οι πρόσθετες DBs χρησιμοποιήθηκαν για την εκπαίδευση των προκαθορισμένων διαμορφώσεων DNN και τον έλεγχο της απόδοσής τους. Όσον αφορά τα υποκατάστατα RBFN, από κάθε διαφορετική DB εκπαιδεύτηκε ένα RBFN δύο εξόδων.

Επιπλέον, άλλα 300 δείγματα, από τις 4 αβέβαιες μεταβλητές, δημιουργήθηκαν τυχαία (ακολουθώντας την κανονική κατανομή) και αξιολογήθηκαν χρησιμοποιώντας τόσο τα υποκατάστατα όσο και τον επιλύτη RANS υψηλής πιστότητας, με στόχο να εκτιμηθεί η απόδοση κάθε εκπαιδευμένου μοντέλου σε αυτή τη μεγαλύτερη αθέατη DB.

Αρχικά βρέθηκε χειροκίνητα μέσω δοκιμής και σφάλματος μια διαμόρφωση DNN καλής απόδοσης. Αποφασίστηκε να δημιουργηθούν δύο ξεχωριστά DNN για κάθε συνθήκη ροής, το ένα για την πρόβλεψη του C_L και το άλλο του C_D. Όλα τα μοντέλα εκπαιδεύτηκαν στο TensorFlow χρησιμοποιώντας το model checkpoint callback για την αποθήκευση του μοντέλου που παρήγαγε τη χαμηλότερο σφάλμα επικύρωσης (validation loss) από όλες τις εποχές, ώστε να αποφευχθεί η υπερπροσαρμογή. Κάθε διαμόρφωση DNN ρυθμίστηκε λεπτομερώς όσον αφορά το F1 και στη συνέχεια εκπαιδεύτηκε και στο F2. Η φάση εκπαίδευσης πραγματοποιήθηκε για 1500 εποχές και, όσον αφορά το υπολογιστικό κόστος, κάθε μοντέλο εκπαιδεύτηκε σε μια GPU

RTX3060 για λιγότερο από ένα λεπτό. Έτσι, το υπολογιστικό κόστος για την εκπαίδευση των δύο μοντέλων είναι λιγότερο από δύο λεπτά. Λαμβάνοντας υπόψη ότι το κόστος δημιουργίας της DB του 40TP, με τη χρήση του επιλυτή RANS, είναι περίπου 10 ώρες, στο συγκεκριμένο σενάριο, το κόστος εκπαίδευσης των DNN μπορεί να θεωρηθεί αμελητέο. Επιπλέον, ο διαχωρισμός επικύρωσης (validation split) επιλέχθηκε στο 0,1, πράγμα που σημαίνει ότι το 90% κάθε DB χρησιμοποιήθηκε για εκπαίδευση και το 10% για επικύρωση. Το μέγεθος της παρτίδας (batch size) ρυθμίστηκε πάντοτε έτσι ώστε να αντιστοιχεί σε ολόκληρη τη DB, καθώς παρατηρήθηκε ότι οι αλλαγές σε αυτή την παράμετρο είχαν ελάχιστο αντίκτυπο στην ακρίβεια του μοντέλου, όπου μάλλον, αυτό οφειλόταν στα μικρά μεγέθη των DBs.

Στη συνέχεια, αξιοποιήθηκε το λογισμικό βελτιστοποίησης με την παροχή της διαμόρφωσης που βρέθηκε με το χέρι ως αρχική υποψήφια λύση. Ο στόχος ήταν να βρεθούν, στον ίδιο αριθμό εποχών, αρχιτεκτονικές DNN που θα παρήγαγαν το ελάχιστο σφάλμα επικύρωσης. Ως εκ τούτου, το σφάλμα επικύρωσης (που παρακολουθείται από το model checkpoint) ορίστηκε ως η αντικειμενική συνάρτηση αυτής της βελτιστοποίησης υπερπαραμέτρων.

Οι μεταβλητές σχεδιασμού ήταν: οι τύποι συναρτήσεων ενεργοποίησης, ένας για κάθε κρυφό στρώμα και ένας για το στρώμα εξόδου, ο αριθμός των κρυφών στρωμάτων και ο αριθμός των νευρώνων ανά στρώμα ως δύναμη του 2. Η διαμόρφωση που βρέθηκε χειροκίνητα παρουσιάζεται δίπλα στην EASY-ευρεθείσα στους πίνακες 7.4 και 7.5. Και οι δύο διαμορφώσεις παρείχαν καλή ικανότητα γενίκευσης. Ωστόσο, η αρχιτεκτονική που βρέθηκε από τον EASY ήταν καλύτερη από την αρχιτεκτονική που βρέθηκε με το χέρι σε κάθε μετρική σφάλματος. Αυτό επιβεβαιώνεται παρατηρώντας τα σφάλματα δοκιμής (test losses) τους στους πίνακες 7.6 και 7.7.

Table 7.4: *NLF(1)-0416 αεροτομή. Διαμόρφωση DNN που βρέθηκε χειροκίνητα*

Layers	Neurons	Activation Function	Batch Size	Loss
7	(4, 128, 64, 32, 64, 1)	ReLU/tanh	Training Patterns	MAE

Table 7.5: *NLF(1)-0416 αεροτομή. Διαμόρφωση DNN που βρέθηκε από τον EASY*

Layers	Neurons	Activation Function	Batch Size	Loss
8	(4, 256, 512, 4096, 64, 32, 512, 1)	GELU/RelU	Training Patterns	MAE

Table 7.6: *NLF(1)-0416 αεροτομή. Σφάλμα δοκιμής DNN (διαμόρφωση που βρέθηκε χειροκίνητα)*

DNN	Test Loss (MAPE)
F1-C_L	0.059%
F1-C_D	0.38%
F2-C_L	0.085%
F2-C_D	0.89%

Table 7.7: *NLF(1)-0416 αεροτομή. Σφάλμαδοκιμής DNN (διαμόρφωση που βρέθηκε από τον EASY)*

DNN	Test Loss (MAPE)
F1-C_L	0.021%
F1-C_D	0.2%
F2-C_L	0.015%
F2-C_D	0.5%

Επιλέγοντας τη EASY-ευρεθείσα διαμόρφωση για τα επόμενα βήματα, τα DNNs και RBFNs εκπαιδεύτηκαν σε όλες τις προαναφερθείσες DBs. Η απόδοση των μοντέλων ελέχθηκε στη DB-300. Τα αποτελέσματα παρουσιάζονται στο σχήμα 7.8.

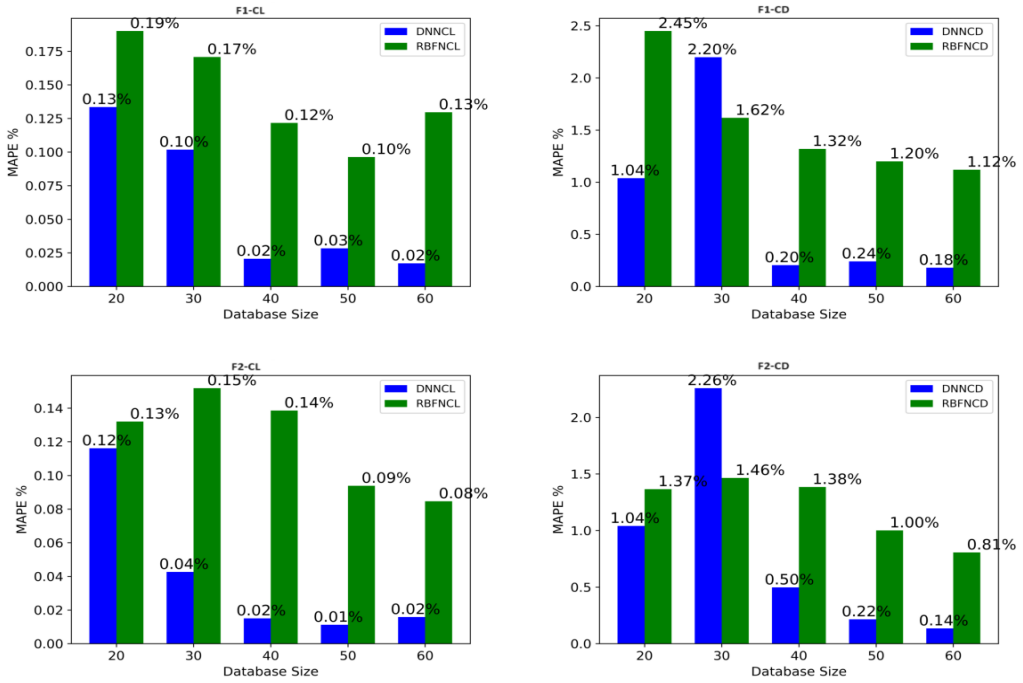

Figure 7.8: *NLF(1)-0416 αεροτομή. Σφάλματα δοκιμής DNN vs. RBFN. Όλα τα μοντέλα κάθε υποπερίπτωσης εκπαιδεύτηκαν στις πέντε προαναφερθείσες ΒΔ και δοκιμάστηκαν στην ίδια DB με 300 δείγματα. Τα πάνω διαγράμματα αφορούν το F1 και τα κατώ το F2. Η υπεροχή των προβλέψεων του DNN είναι προφανής.*

Στη συνέχεια, χρησιμοποιώντας τα DNN και το CFD, πραγματοποιήθηκε UQ με MC, gPCE και rPCE για αρκετούς αριθμούς δειγμάτων. Τα πιο σημαντικά στοιχεία από αυτή τη μελέτη UQ συνοψίζονται στους πίνακες 7.8 και 7.9, για τις F1 και F2.

Table 7.8: *NLF(1)-0416 αεροτομή. Μέθοδοι UQ για F1*

Method/Tool	**Time Units**	μC_L	σC_L	μC_D	σC_D
MC-DNN(500)	40	0.7208	0.004933	0.006179	0.0004484
gPCE-DNN($k = 2,\ 81$)	40	0.7210	0.005002	0.006161	0.0004573
rPCE-DNN($k = 2,\ 140$)	40	0.7204	0.004310	0.006236	0.0004196
MC-CFD(300)	300	0.7208	0.004904	0.006185	0.0004792
gPCE-CFD($k = 2,\ 81$)	81	0.7210	0.004923	0.006153	0.0004477
rPCE-CFD($k = 2,\ 81$)	81	0.7208	0.004584	0.006174	0.0004199

Table 7.9: *NLF(1)-0416 αεροτομή. Μέθοδοι UQ για F2*

Method/Tool	**Time Units**	μC_L	σC_L	μC_D	σC_D
MC-DNN(500)	40	1.0206	0.006888	0.007669	0.0005546
gPCE-DNN($k = 2,\ 81$)	40	1.0211	0.006978	0.007634	0.0005648
rPCE-DNN($k = 2,\ 140$)	40	1.0207	0.005712	0.007649	0.0004329
MC-CFD(300)	300	1.0206	0.00705	0.007632	0.0005473
gPCE-CFD($k = 2,\ 81$)	40	1.0210	0.006955	0.007603	0.0005399
rPCE-CFD($k = 2,\ 81$)	81	1.0209	0.00634	0.007608	0.0004929

Τέλος, πραγματοποιήθηκαν δύο πρόσθετες μελέτες σχετικά με το υποκατάστατο μοντέλο. Πρώτον, για την αύξηση της ακρίβειας, τα DNN και τα RBFN συνδυάστηκαν μέσω της τεχνικής stacking ensemble με τη χρήση ενός μοντέλου γραμμικής παλινδρόμησης (Linear Regression) ως μέτα-μοντέλο (τα αποτελέσματα απεικονίζονται στο σχήμα 7.9).

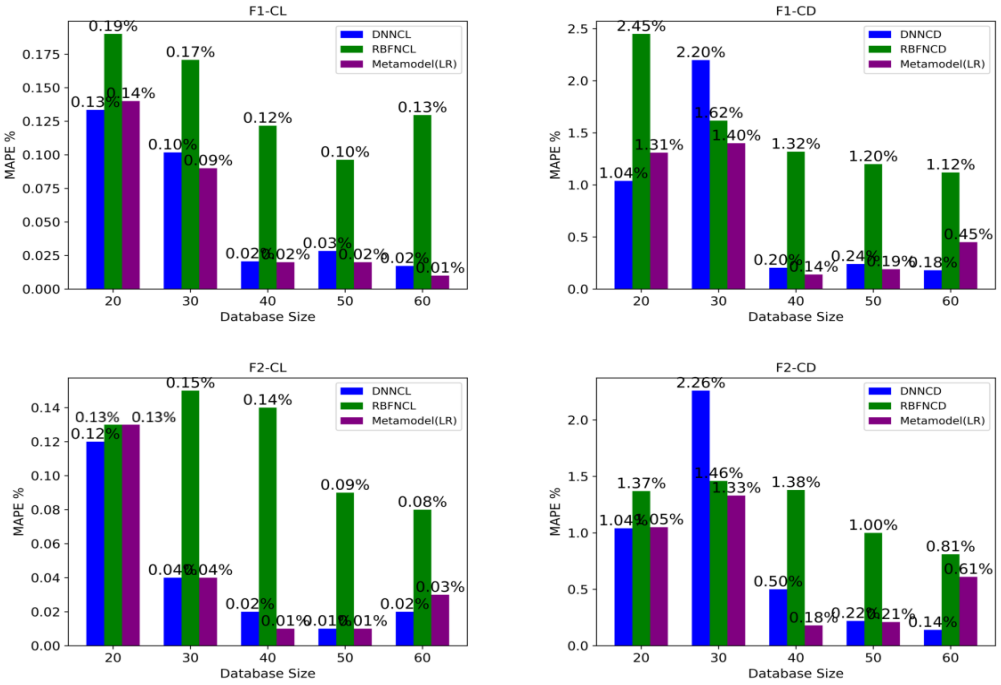

Figure 7.9: *NLF(1)-0416 αεροτομή. Σφάλματα δοκιμής DNN vs. RBFN vs. Metamodel (LR). Όλα εκπαιδεύτηκαν στις πέντε παραπάνω DBs και δοκιμάστηκαν στην ίδια ΒΔ 300 δειγμάτων. Τα πάνω διαγράμματα αφορούν το F1 και τα κατώ το F2. Στις περισσότερες περιπτώσεις (περίπου 3 στις 5), το συνδυαστικό μοντέλο παρείχε βελτιωμένη ακρίβεια. Ωστόσο, σε ορισμένες άλλες περιπτώσεις, είχε χειρότερες επιδόσεις από το DNN.*

Δεύτερον, ακολουθώντας τις υποδείξεις του PCC (που εξηγείται στο Παράρτημα Α.1.3), τα DNN εκπαιδεύτηκαν αποκλειστικά στην DB-40, αφήνοντας εκτός τα πιθανώς λιγότερο σημαντικά χαρακτηριστικά εισόδου (σχήμα 7.10. Πραγματοποιήθηκαν τρεις διαφορετικές δοκιμές:

- Test1: Εκπαίδευση DNN παραλείποντας c_{a1}
- Test2: Εκπαίδευση DNN παραλείποντας $c_{\theta,t}$
- Test3: Εκπαίδευση DNN παραλείποντας c_{a1} και $c_{\theta,t}$

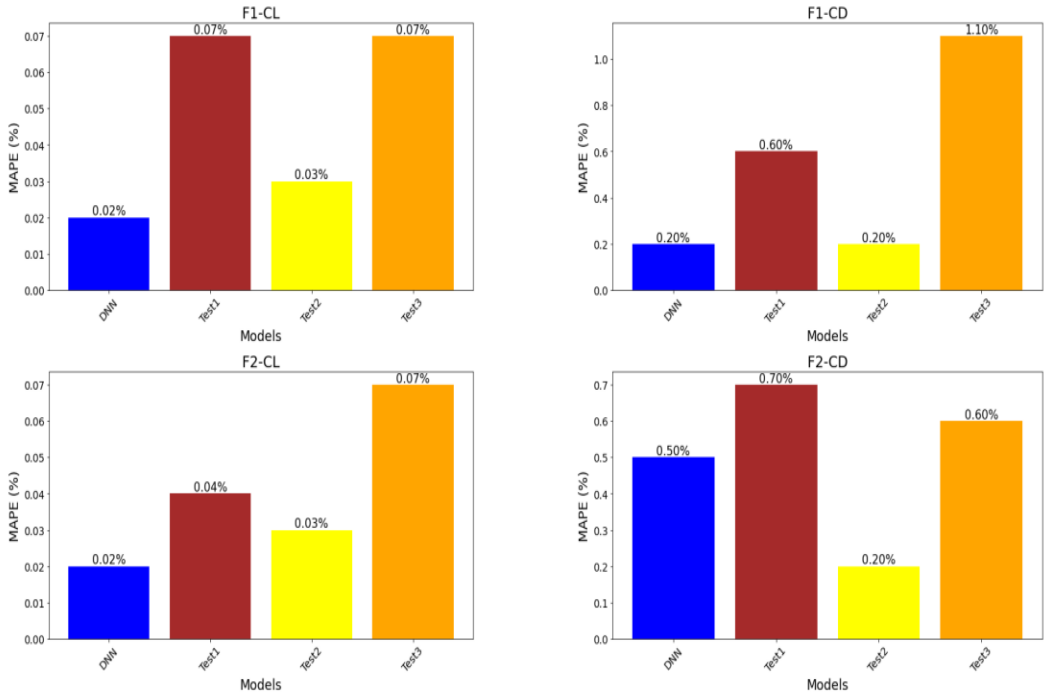

Figure 7.10: *NLF(1)-0416 αεροτομή. Το αρχικό σφάλμα δοκιμής DNN συγκρίνεται με αυτά των Test1, Test2 και Test3 για F1 (άνω διαγράμματα) και F2 (κάτω διαγράμματα).*

Ποσοτικοποίηση Αβεβαιότητας στην μεμονωμένη πτέρυγα ONERA M6

Αυτή η εφαρμογή ασχολείται με τη χρήση DNNs για UQ στην περίπτωση μιας τριδιάστατης ροής γύρω από μια μεμονωμένη πτέρυγα. Πέραν των τεσάρων σταθερών του μοντέλου μετάβασης, η τραχύτητα της επιφάνειας, $h_{rms} \sim \mathcal{N}(5 \cdot 10^{-6}, 1.6 \cdot 10^{-6})$, συμπεριλήφθηκε ως πέμπτη αβέβαιη μεταβλητή (M=5). Αυτή η ποσότητα μπορεί επίσης να χαρακτηριστεί αμφισβητήσιμης πιστότητας, κυρίως λόγω περιορισμών των μετρήσεων. Στόχος ήταν να ποσοτικοποιηθεί η επίδραση των αβεβαιοτήτων στην C_L και στην C_D, χρησιμοποιώντας DNN.

Το πρωταρχικό πρόβλημα αφορά μια ροή γύρω από την πτέρυγα ONERA M6 με $M_\infty = 0.262$, $Re = 3.5 \cdot 10^6$, γωνία προσβολής και γωνία εκτροπής και οι δύο ρυθμισμένες σε 0^o. Η ένταση της τύρβης είναι Tu=0.2%. Τα διαθέσιμα προ-αξιολογημένα 120 αβέβαια διανύσματα λήφθηκαν με χρήση LHS. Τα δεδομένα οργανώθηκαν σε δύο DBs: η μία με 80 TP και η δεύτερη με όλα τα διαθέσιμα TP, δηλαδή 120.

Οι στατιστικές ροπές του C_f τόσο για την πλευρά της υποπίεσης όσο και για την πλευρά της πίεσης υπολογίστηκαν με rPCE-CFD. Και για τις δύο πλευρές της πτέρυγας τα αποτελέσματα ήταν σχεδόν πανομοιότυπα, και, επομένως, μόνο αυτά της πλευράς αναρρόφησης απεικονίζονται στα Σχήματα 7.11. Σε αυτά φαίνεται το πού λαμβάνει

χώρα κατά μέσο όρο η έναρξη της μετάβασης, παράλληλα με το πόσο ευαίσθητη είναι στις εξεταζόμενες αβεβαιότητες.

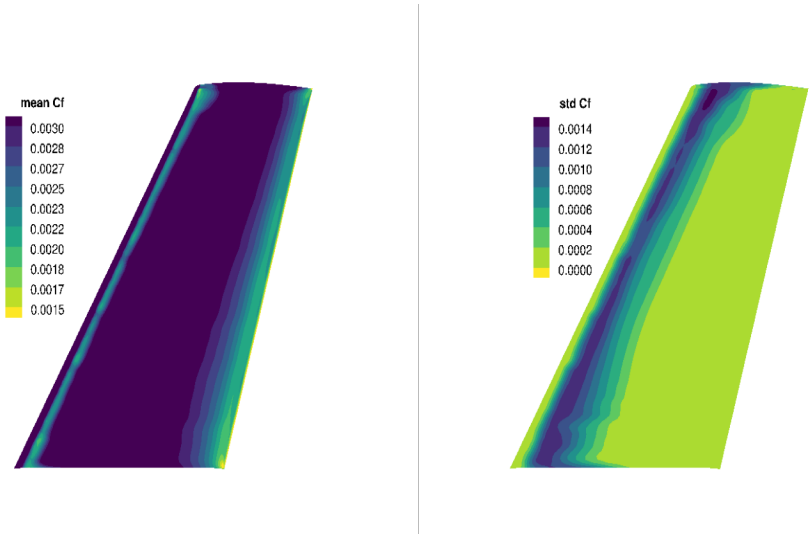

Figure 7.11: *ONERA M6 πτέρυγα. Στατιστικές ροπές του C_f, πλευρά πίεσης. Μέση τιμή (αριστερά) και τυπική απόκλιση (δεξιά).*

Η rPCE διεξήχθη για τα DB-80 και DB-120 και τα αποτελέσματα θεωρήθηκαν ως τα αποτελέσματα UQ υψηλής πιστότητας.

Για κάθε DB, χρησιμοποιήθηκε το 90% των δεδομένων για εκπαίδευση και το 10% για επικύρωση. Πραγματοποιήθηκαν δύο προσεγγίσεις. Θεωρώντας το C_D/C_L ως QoI, η πρώτη προσέγγιση περιελάμβανε ένα DNN μίας εξόδου, το οποίο προέβλεπε απευθείας την QoI. Η δεύτερη προσέγγιση πρότεινε, όπως και στην προηγούμενη περίπτωση, τη χρήση δύο DNN μίας εξόδου. Το ένα για την πρόβλεψη του C_L και το άλλο για την πρόβλεψη του C_D. Λαμβάνοντας υπόψη ότι οι δύο προσεγγίσεις αφορούσαν συνολικά τρία μοντέλα και ότι ο στόχος ήταν να ελεγχθούν οι επιδόσεις των μοντέλων που εκπαιδεύτηκαν στις δύο DBs, 6 είναι ο αριθμός των μοντέλων που θα έπρεπε να εκπαιδευτούν και να χρησιμοποιηθούν για την UQ. Αυτή τη φορά, δεν αξιοποιήθηκε κανένα σύνολο δοκιμαστικών δεδομένων. Κάθε DNN χρησιμοποιήθηκε απευθείας για UQ.

Αρχικά, η χειροκίνητα δημιουργηθείσα διαμόρφωση, 7.4, από την περίπτωση I αξιολογήθηκε για τη δημιουργία των 6 DNN και στη συνέχεια, ο EASY χρησιμοποιήθηκε για 6 βελτιστοποιήσεις υπερπαραμέτρων, δηλαδή μία για κάθε επιθυμητό μοντέλο. Παρά την εκτενή βελτιστοποίηση των υπερπαραμέτρων, τα αποτελέσματα UQ των υποκατάστατων που βρέθηκαν με EASY ήταν, μάλλον, ελάχιστα ανώτερα και παρουσιάζονται δίπλα στις αναφορές (rPCE-CFD) στους πίνακες 7.10 και 7.11.

Table 7.10: *ONERA M6 πτέρυγα. Συγκεντρωτικά αποτελέσματα UQ με χρήση DB-80:*

Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}
rPCE-CFD(80)	80	6.8636	0.3180
rPCE-DNN(80)	80	6.8620	0.3100
rPCE-DNN(120)	80	6.8803	0.3238
MC-DNN(50^4)	80	6.9528	0.4020
gPCE-DNN(243)	80	6.9523	0.3908

Table 7.11: *ONERA M6 πτέρυγα. Συγκεντρωτικά αποτελέσματα UQ με χρήση DB-120:*

Method/Tool	**Time Units**	μ_{C_D/C_L}	σ_{C_D/C_L}
rPCE-CFD(120)	120	6.8571	0.3278
rPCE-DNN(80)	120	6.8463	0.3447
rPCE-DNN(120)	120	6.8950	0.3407
MC-DNN(50^4)	80	6.9528	0.4020
gPCE-DNN(243)	120	6.9167	0.3879
MC-DNN(20^4)	120	6.9180	0.39180

Τέλος, 40 από τα 120TP χρησιμοποιήθηκαν για να ελεχθεί η προβλεπτική απόδοση δύο επιπλέον ML μοντέλων: K-Nearest Neighbors και Support Vector Regression δίπλα στα DNN (σχήμα 7.12). Έτσι, το DB-80 χρησιμοποιήθηκε για την εκπαίδευση των μοντέλων και η DB-40 χρησιμοποιήθηκε αποκλειστικά για τη δοκιμή τους.

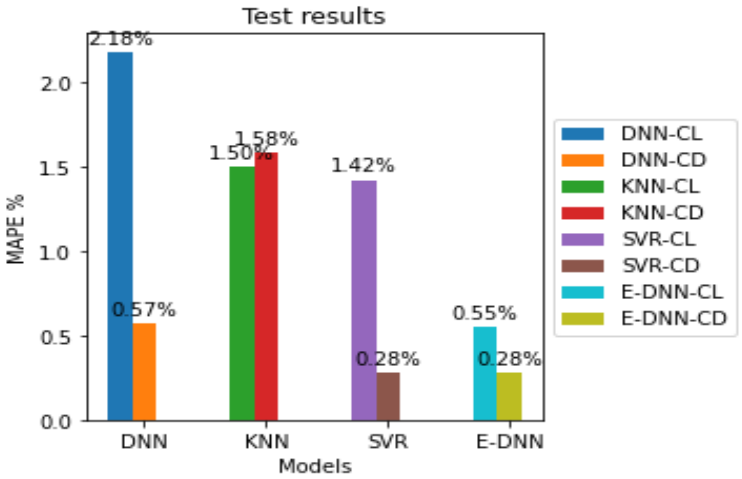

Figure 7.12: *ONERA M6 πτέρυγα. Τα μοντέλα που βρέθηκαν με το λογισμικό EASY συμβολίζονται ως «E-DNN». Τόσο το KNN όσο και το SVR προβλέπουν πολύ ικανοποιητικά τις QoIs. Ειδικότερα, το SVR υπερτερεί του DNN στην πρόβλεψη των CL και CD, ενώ το KNN-CL υπερτερεί του DNN-CL. Τα E-DNN είναι ανώτερα και ίσα με τα SVR στην πρόβλεψη των CL και CD, αντίστοιχα.*

Συμπεράσματα

Οι διακυμάνσεις στις αβέβαιες μεταβλητές όσον αφορά και τις δύο εφαρμογές παρατηρείται ότι μετατοπίζουν σημαντικά το σημείο μετάβασης, αναδεικνύοντας τον σοβαρό αντίκτυπο των αβεβαιοτήτων που εμπλέκονται. Τα DNN μπορούσαν να προβλέψουν τα C_L και C_D των μεταβατικών ροών που αξιολογήθηκαν με επαρκή ακρίβεια. Όπως αποδείχθηκε στην περίπτωση της αεροτομής, η χρήση τους μπορεί να μειώσει το υπολογιστικό κόστος, της UQ, πάνω από 50% ανά υποψήφια λύση. Η χρήση εξελικτικών αλγορίθμων για τη βελτιστοποίηση των υπερπαραμέτρων των DNN είναι μια πρακτική που μπορεί να αυξήσει σημαντικά την ακρίβεια πρόβλεψης (έως και 82% βελτίωση).

Στην περίπτωση της αεροτομής, τα DNN αποδείχθηκαν έως και 10 φορές πιο ακριβή από τα RBFN, αλλά ήταν πιο πολύπλοκα και περίπου 10 φορές πιο ακριβά για να εκπαιδευτούν. Επιπλέον, ο συνδυασμός των δύο τελευταίων μέσω της τεχνικής stacking ensemble παρείχε μοντέλα που ήταν έως και 50% ακριβέστερα από τα DNN. Το αντιστάθμισμα είναι ότι το συνδυαστικό μοντέλο περιλαμβάνει το κόστος κάθε συνδυαζόμενου μοντέλου, τα οποία στην παρούσα περίπτωση ήταν αμελητέα.

Η εκπαίδευση των DNN χρησιμοποιώντας feature selection (περίπτωση αεροτομής) δημιούργησε απλούστερα μοντέλα (με έως και 20% ταχύτερο χρόνο εκπαίδευσης) τα οποία είχαν παρόμοια επίδοση με το αρχικό. Επίσης, ένα από τα δώδεκα DNN με μειωμένες εισόδους, αποδείχθηκε 60% καλύτερο από το αρχικό στην πρόβλεψη του C_D (Test2).

Η gPCE αποδείχθηκε ακριβέστερη από την rPCE, γεγονός που είναι λογικό αφού υπόκειται στην κατάρα των πολλαπλών διαστάσεων, ενώ η rPCE όχι. Επιπλέον, όπως φάνηκε στην περίπτωση της πτέρυγας, όταν μια μέθοδος UQ απαιτεί περισσότερες κλήσεις στο υποκατάστατο, η ακρίβειά της κινδυνεύει να μειωθεί λόγω της διευρυμένης διάδοσης των σφαλμάτων του υποκατάστατου.

Τα KNN και SVR ήταν επίσης ικανά να χρησιμοποιηθούν ως υποκατάστατα (περίπτωση πτέρυγας), παρέχοντας έως και 51% καλύτερες προβλέψεις από κάποια DNN. Η εκπαίδευσή τους ήταν κατ' ελάχιστον 5 φορές ταχύτερη, και αξίζει να δοκιμάζονται πριν τη χρήση εξελικτικών αλγορίθμων για βελτιστοποίηση υπερπαραμέτρων DNN.